

**The USENIX Association**

**Proceedings of the  
Second USENIX Workshop on Electronic  
Commerce**

**Co-Sponsored by**

**The Fisher Center for Information Technology Management, UC Berkeley  
and the School of Information Management and Systems, UC Berkeley**

**November 18-21, 1996  
Oakland, California**

**Program Chair**

Doug Tygar, *Carnegie Mellon University*

**Program Committee**

Ross Anderson, *Cambridge University*

Nathaniel Borenstein, *First Virtual Holdings*

Stefan Brands, *CWI*

Daniel Geer, *Open Market, Inc.*

Mark Manasse, *Digital Equipment Corporation*

Clifford Neuman, *University of Southern California*

Hal Varian, *University of California, Berkeley*

Bennet Yee, *University of California, San Diego*

# CONTENTS

Preface .....	v
Author Index .....	vi

## Tuesday, November 19

### Hardware Tokens

Tamper Resistance-a Cautionary Note .....	1
<i>Ross Anderson, Cambridge University; Markus Kuhn, Purdue University</i>	

Token-Mediated Certification and Electronic Commerce .....	13
<i>Daniel E. Geer, Donald T. Davis, Open Market, Inc.</i>	

Smart Cards in Hostile Environments .....	23
<i>Howard Gobioff, Carnegie Mellon University; Sean Smith, IBM Research; J.D. Tygar, Carnegie Mellon University; Bennet Yee, UC San Diego</i>	

### Protocol Analysis

Analysis of the SSL 3.0 Protocol .....	29
<i>David Wagner, University of California, Berkeley; Bruce Schneier, Counterpane Systems</i>	

Fast, Automatic Checking of Security Protocols .....	41
<i>Darrell Kindred, Jeannette Wing, Carnegie Mellon University</i>	

Verifying Cryptographic Protocols for Electronic Commerce .....	53
<i>Randall W. Lichota, Hughes; Grace L. Hammonds, AGCS, Inc.; Stephen H. Brackin, Arca Systems, Inc.</i>	

<b>Invited Talk: Legal Signatures and Proof in Electronic Commerce .....</b>	67
<i>Benjamin Wright, Attorney and Author: The Law of Electronic Commerce</i>	

### Policy and Economics

Digital Currency and Public Networks: So What If It Is Secure, Is It Money? .....	77
<i>John du Pre Gauntt, London School of Economics</i>	

Modeling the Risks and Costs of Digitally Signed Certificates in Electronic Commerce .....	287
<i>Ian Simpson, Carnegie Mellon University</i>	

### Standard Payment Interfaces

Generic Electronic Payment Services: Framework and Functional Specification .....	87
<i>Alireza Bahreman, EIT</i>	

U-PAI: A Universal Payment Application Interface .....	105
<i>Steven P. Ketchpel, Hector Garcia-Molina, Andreas Paepcke, Scott Hassan, Steve Cousins, Stanford University</i>	

Payment Method Negotiation Service: Framework and Programming Interface .....	299
<i>Alireza Bahreman, Rajkumar Narayanaswamy, EIT</i>	

## Wednesday, November 20

### Atomic Transactions

Anonymous Atomic Transactions .....	123
<i>Jean Camp, Sandia National Laboratory; Michael Harkavy, J.D. Tygar, Carnegie Mellon University; Bennet Yee, University of California, San Diego</i>	

Strongboxes for Electronic Commerce .....	135
<i>Thomas Hardjono, Jennifer Seberry, University of Wollongong</i>	
Model Checking Electronic Commerce Protocols.....	147
<i>Nevin Heintze, Bell Labs; J.D. Tygar, Jeannette Wing, H. Chi Wong, Carnegie Mellon University</i>	
<b>Experience</b>	
BigDog: Hierarchical Authentication, Session Control, and Authorization for the Web .....	165
<i>Benjamin Fried, Andrew Lowry, Morgan Stanley</i>	
Financial EDI Over the Internet: Case Study II.....	173
<i>Arie Segev, Jaana Porra, Malu Roldan, University of California, Berkeley</i>	
Scalable Document Fingerprinting .....	191
<i>Nevin Heintze, Bell Labs</i>	
<b>Protocols</b>	
A Protocol for Secure Transactions .....	201
<i>Douglas H. Steves, Chris Edmondson-Yurkanan, Mohamed Gouda, University of Texas, Austin</i>	
PayTree: "Amortized-Signature" for Flexible MicroPayments .....	213
<i>Charanjit Jutla, IBM, Moti Yung, Bankers Trust</i>	
Agora: A Minimal Distributed Protocol for Electronic Commerce.....	223
<i>Eran Gabber, Abraham Silberschatz, Bell Labs</i>	
<b>Thursday, November 21</b>	
<b>Security</b>	
Organizing Electronic Services into Security Taxonomies .....	233
<i>Sean Smith, IBM Research; Paul Pedersen, Los Alamos National Laboratory</i>	
WWW Electronic Commerce and Java Trojan Horses .....	243
<i>J.D. Tygar, Alma Whitten, Carnegie Mellon Univiristy</i>	
On Shopping Incognito .....	251
<i>Ralf Hauser, McKinsey Consulting, Switzerland; Gene Tsudik, University of Southern California</i>	
<b>Software Agents</b>	
Market-Based Negotiation for Digital Library Services .....	259
<i>Tracy Mullen, Michael P. Wellman, University of Michigan, Ann Arbor</i>	
Information and Interaction in MarketSpace--Towards an Open Agent-based Market Infrastructure .....	271
<i>Joakim Eriksson, Niclas Finne, Sverker Janson, Swedish Institute of Computer Science</i>	
A Peer-to-Peer Software Metering System .....	279
<i>Bruce Schneier, John Kelsey, Counterpane Systems</i>	

# Preface

Sixteen months have passed since the First USENIX Workshop on Electronic Commerce—and what a difference those sixteen months have made! A large number of new electronic commerce systems have been released; many more are in final stages of preparation, and the industry has seen active movement to try to establish standards. Along with the rapid growth in new electronic commerce systems, we have seen a parallel growth in electronic commerce meetings and forums. But there is still a special place for this workshop. The USENIX Electronic Commerce Workshop continues its charter as a scholarly, scientific forum for all aspects of electronic commerce: engineering, business, theoretical, and systems issues.

This year, we have 26 refereed papers (out of 53 submissions) as well as an invited paper by Ben Wright. We hope to prepare a digest of the meeting containing a summary of the talks, questions, and discussion. This digest will be available from the URL <http://www.usenix.org/publications/library/index.html> along with electronic versions of the submitted papers. In addition, we expect to publish the digest in an issue of the USENIX newsletter; *login*. We also hope that it will appear in the proceedings of the next USENIX Workshop on Electronic Commerce. I sincerely thank the program committee for their work in reviewing papers and the USENIX staff for their excellent work in putting this workshop together. I especially thank the Fisher Center for Information Technology Management and the School of Information Management, both at UC Berkeley, for co-sponsoring this workshop.

May all your electronic transactions be secure and atomic!

Doug Tygar  
Program Chair

## Author Index

Ross Anderson . . . . .	1
Alireza Bahreman . . . . .	87, 299
Stephen H. Brackin . . . . .	53
Jean Camp . . . . .	123
Steve Cousins . . . . .	105
Donald T. Davis . . . . .	13
Chris Edmondson-Yurkanan . . . . .	201
Joakim Eriksson . . . . .	271
Niclas Finne . . . . .	271
Benjamin Fried . . . . .	165
Eran Gabber . . . . .	223
Hector Garcia-Molina . . . . .	105
Daniel E. Geer . . . . .	13
Howard Gobioff . . . . .	23
Mohamed Gouda . . . . .	201
Grace L. Hammonds . . . . .	53
Thomas Hardjono . . . . .	135
Michael Harkavy . . . . .	123
Scott Hassan . . . . .	105
Ralf Hauser . . . . .	251
Nevin Heintze . . . . .	147, 191
Sverker Janson . . . . .	271
Charanjit Jutla . . . . .	213
John Kelsey . . . . .	279
Steven P. Ketchpel . . . . .	105
Darrell Kindred . . . . .	41
Markus Kuhn . . . . .	1
Randall W. Lichota . . . . .	53
Andrew Lowry . . . . .	165
Tracy Mullen . . . . .	259
Rajkuman Narayanaswamy . . . . .	299
Andreas Paepcke . . . . .	105
Paul Pedersen . . . . .	233
Jaana Porra . . . . .	173
John du Pre Gauntt . . . . .	77
Malu Roldan . . . . .	173
Bruce Schneier . . . . .	29, 279
Jennifer Seberry . . . . .	135
Arie Segev . . . . .	173
Abraham Silberschatz . . . . .	223
Ian Simpson . . . . .	287
Sean Smith . . . . .	23, 233
Douglas H. Steves . . . . .	201
Gene Tsudik . . . . .	251
J.D. Tygar . . . . .	23, 123, 147, 243
David Wagner . . . . .	29
Michael P. Wellman . . . . .	259
Alma Whitten . . . . .	243
Jeannette Wing . . . . .	41
H. Chi Wong . . . . .	147
Benjamin Wright . . . . .	67
Bennet Yee . . . . .	123
Moti Yung . . . . .	213

# Tamper Resistance — a Cautionary Note

Ross Anderson

*Cambridge University  
Computer Laboratory  
Pembroke Street  
Cambridge CB2 3QG  
England  
rja14@cl.cam.ac.uk*

Markus Kuhn

*COAST Laboratory  
Department of Computer Sciences  
Purdue University  
West Lafayette, IN 47907  
U.S.A.  
kuhn@cs.purdue.edu*

## Abstract

*An increasing number of systems, from pay-TV to electronic purses, rely on the tamper resistance of smartcards and other security processors. We describe a number of attacks on such systems — some old, some new and some that are simply little known outside the chip testing community. We conclude that trusting tamper resistance is problematic; smartcards are broken routinely, and even a device that was described by a government signals agency as ‘the most secure processor generally available’ turns out to be vulnerable. Designers of secure systems should consider the consequences with care.*

## 1 Tamperproofing of cryptographic equipment

Many early cryptographic systems had some protection against the seizure of key material. Naval code books were weighted; rotor machine setting sheets were printed using water soluble ink; and some one-time pads were printed on cellulose nitrate, so that they would burn rapidly if lit [20].

But such mechanisms relied on the vigilance of the operator, and systems were often captured in surprise attacks. So cryptographic equipment designed in recent years has often relied on technical means to prevent tampering. An example is the VISA security module, commonly used in banks to generate and check the personal identification numbers (PINs) with which customers authenticate themselves at automatic teller machines. It is basically a safe containing a microcomputer that performs all the relevant cryptographic operations; the safe has lid switches and circuitry which interrupts power to memory, thus erasing key material, when the lid is opened [27]. The idea is to deny the bank’s programmers access to customer PINs and the keys

that protect them; so when a customer disputes a transaction, the bank can claim that the customer must have been responsible as no member of its staff had access to the PIN [6].

Evaluating the level of tamper resistance offered by a given product is thus an interesting and important problem, but one which has been neglected by the security research community. One of the few recent articles that discuss the subject describes the design of the current range of IBM products and proposes the following taxonomy of attackers [1]:

**Class I (clever outsiders):** They are often very intelligent but may have insufficient knowledge of the system. They may have access to only moderately sophisticated equipment. They often try to take advantage of an existing weakness in the system, rather than try to create one.

**Class II (knowledgeable insiders):** They have substantial specialized technical education and experience. They have varying degrees of understanding of parts of the system but potential access to most of it. They often have highly sophisticated tools and instruments for analysis.

**Class III (funded organisations):** They are able to assemble teams of specialists with related and complementary skills backed by great funding resources. They are capable of in-depth analysis of the system, designing sophisticated attacks, and using the most advanced analysis tools. They may use Class II adversaries as part of the attack team.

The critical question is always whether an opponent can obtain unsupervised access to the device [22]. If the answer is no, then relatively simple measures may suffice. For example, the VISA security

module is vulnerable to people with occasional access: a service engineer could easily disable the tamper protection circuitry on one of her visits, and extract key material on the next. But this is not considered to be a problem by banks, who typically keep security modules under observation in a computer room, and control service visits closely.

But in an increasing number of applications, the opponent can obtain completely unsupervised access, and not just to a single instance of the cryptographic equipment but to many of them. This is the case that most interests us: it includes pay-TV smartcards, prepayment meter tokens, remote locking devices for cars and SIM cards for GSM mobile phones [4]. Many such systems are already the target of well funded attacks.

So in what follows, we will assume that all attackers can obtain several examples of the target equipment. We will also ignore tampering at the circuit board level (though this has caused losses, for example, with prepaid electricity meters [7]) and rather concentrate on attacks aimed at recovering crypto key material stored in smartcards and other chip-level security processors.

## 2 Breaking smartcards and microcontrollers

The typical smartcard consists of an 8-bit microprocessor with ROM, EEPROM and RAM, together with serial input and output, all in a single chip that is mounted on a plastic carrier. Key material is kept in the EEPROM.

Designers of EEPROM based devices face a problem: erasing the charge stored in the floating gate of a memory cell requires a relatively high voltage. If the attacker can remove this, then the information will be trapped.

Early smartcards received their programming voltage on a dedicated connection from the host interface. This led to attacks on pay-TV systems in which cards were initially enabled for all channels, and those channels for which the subscriber did not pay were deactivated by broadcast signals. By covering the programming voltage contact on their card with tape, or by clamping it inside the decoder using a diode, subscribers could prevent these signals affecting the card. They could then cancel their subscription without the vendor being able to cancel their service.

Some cards are still vulnerable to this kind of attack, and it gives rise to a sporadic failure mode of some card-based public telephone systems: telephones where the relevant contact is dirty or bent

may fail to decrement any user's card. However, the cards used nowadays in pay-TV decoders generate the required 12 V from the normal 5 V power supply using an on-chip oscillator and diode/capacitor network. This can push up the cost of an attack, but does not make it impossible: large capacitors can be identified under a microscope and destroyed with lasers, ultrasonics or focused ion beams. A chip prepared in this way can be investigated at will without the risk of erasing the EEPROM.

So our task is to classify the various logical and physical attacks on security processors and get some idea of the cost involved.

### 2.1 Non-invasive attacks

Unusual voltages and temperatures can affect EEPROM write operations. For instance, for the PIC16C84 microcontroller, a trick has become widely known that involves raising VCC to VPP – 0.5 V during repeated write accesses to the security bit. This can often clear it without erasing the remaining memory.

For the DS5000 security processor, a short voltage drop sometimes released the security lock without erasing secret data. Processors like the 8752 that can be used with both internal and external memory but that limit the switch between them to resets have been read out using low voltages to toggle the mode without a reset. Low voltage can facilitate other attacks too: at least one card has an on-board analogue random number generator, used to manufacture cryptographic keys and nonces, which will produce an output of almost all 1's when the supply voltage is lowered slightly.

For these reasons, some security processors have sensors that cause a reset when voltage or other environmental conditions go out of range. But any kind of environmental alarm will cause some degradation in robustness. For example, one family of smartcard processors was manufactured with a circuit to detect low clock frequency and thus prevent single-stepping attacks. However, the wild fluctuations in clock frequency that frequently occur when a card is powered up and the supply circuit is stabilising, caused so many false alarms that the feature is no longer used by the card's operating system. Its use is left to the application programmer's discretion. Few of them bother; those who do try to use it discover the consequences for reliability. So many cards can be single-stepped with impunity.

For similar robustness reasons, the under-voltage and over-voltage detection circuitry in many devices will not react to transients. So fast signals of various

kinds may reset the protection without destroying the protected information, and attacks of this kind are now known in the community for quite a number of devices.

Power and clock transients can also be used in some processors to affect the decoding and execution of individual instructions. Every transistor and its connection paths act like an RC element with a characteristic time delay; the maximum usable clock frequency of a processor is determined by the maximum delay among its elements. Similarly, every flip-flop has a characteristic time window (of a few picoseconds) during which it samples its input voltage and changes its output accordingly. This window can be anywhere inside the specified setup cycle of the flip-flop, but is quite fixed for an individual device at a given voltage and temperature.

So if we apply a clock glitch (a clock pulse much shorter than normal) or a power glitch (a rapid transient in supply voltage), this will affect only some transistors in the chip. By varying the parameters, the CPU can be made to execute a number of completely different wrong instructions, sometimes including instructions that are not even supported by the microcode. Although we do not know in advance which glitch will cause which wrong instruction in which chip, it can be fairly simple to conduct a systematic search.

A typical subroutine found in security processors is a loop that writes the contents of a limited memory range to the serial port:

```
1 b = answer_address
2 a = answer_length
3 if (a == 0) goto 8
4 transmit(*b)
5 b = b + 1
6 a = a - 1
7 goto 3
8 ...
```

We can look for a glitch that increases the program counter as usual but transforms either the conditional jump in line 3 or the loop variable decrement in line 6 into something else.

Finding the right glitch means operating the card in a repeatable way. All signals sent to it have to arrive at exactly the same time after reset for every test run. Many glitches can be tested for every clock cycle, until one of them causes an extra byte to be sent to the serial port. Repeating it causes the loop to dump the remaining memory, which if we are lucky will include the keys we are looking for.

Output loops are just one target for glitch attacks. Others are checks of passwords, access rights and protocol responses, where corruption of a single instruction can defeat the protection. A possible software countermeasure might be to avoid single-point-of failure instructions. This was common enough in the old days of unreliable hardware: a senior Cambridge computer scientist recalls that in the 1950's a prudent system programmer was someone who, having masked off three bits, would verify that the result did not exceed seven!

Hardware countermeasures include independent internal clock generators that are only PLL synchronized with the external reference frequency.

## 2.2 Physical attacks

Physical attacks on some microcontrollers are almost trivial. For example, the lock bit of several devices with on-chip EEPROM can be erased by focusing UV light on the security lock cell, which is located sufficiently far from the rest of memory.

Current smartcards are slightly harder to attack, but not very much harder. They generally have little to prevent direct access to the silicon; the marketing director of a smartcard vendor claimed that there was simply no demand from their users for anything really sophisticated [21]. The most that appears to be done is a capacitive sensor to detect the continued presence of the passivation layer [23], or an optical sensor under an opaque coating [3]. Similar robustness considerations apply to these detectors as to the ones discussed above; they are often not used, and when they are, they are fairly easy to detect and avoid.

Anyway, the typical chip module consists of a thin plastic basis plate of about a square centimetre with conductive contact areas on both sides. One side is visible on the final card and makes contact with the card reader; the silicon die is glued to the other side, and connected using thin gold or aluminium bonding wires. The chip side of the plastic plate is then covered with epoxy resin. The resulting chip module is finally glued into the card, which is available in ISO credit card format, in miniature format for some GSM systems, or in the case of some prepayment electricity meter systems and pay-TV systems resembles a small plastic key.

Removing the chip is easy. First, we use a sharp knife or hand lathe to cut away the plastic behind the chip module until the epoxy resin becomes visible. Now we settle a few drops of fuming nitric acid (>98% HNO<sub>3</sub>) on the resin and wait a few minutes until some of it has dissolved (the process can be

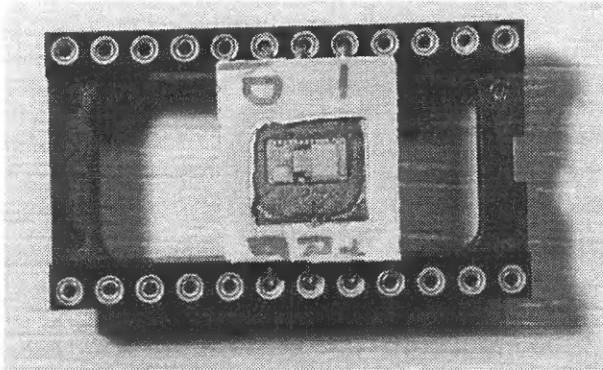


Figure 1: Fully functional smartcard processor with covering plastic removed for microprobing experiments. All tools necessary for this preparation were obtained for US\$30 in a pharmacy.

accelerated by heating up the acid with an infra-red radiator). Before the acid dissolves too much epoxy and gets solid, we wash acid and resin away by shaking the card in acetone. We repeat this procedure around five to ten times until the silicon surface of the die is fully exposed. The chip can then be washed and will be fully functional unless one of the bonding wires has been damaged.

Functional tests with pay-TV and prepaid phone smartcards have shown that EEPROM content is not affected by hot nitric acid. No knowledge beyond school chemistry is necessary; the materials are easily available in any chemistry lab, and several undergraduate students have recently reported the successful application of this method on an Internet mailing list dedicated to amateur smartcard hacking. Fuming nitric acid is an aggressive oxidant and should be handled carefully (especially when using flammable liquids such as acetone), but it does not affect silicon, silicon oxide, silicon nitride, or gold as used on the chip and its contacts. The aluminium used in the metal layer of the chip is covered at once with a thin oxide layer and is also unaffected. Nitric acid is commonly used anyway to clean chip surfaces during manufacture.

There are commercial IC package removal machines used in process quality control, which expose the chip to an  $\text{HNO}_3$  vapor stream that not only dissolves the resin but also transports away the waste products. This leaves a somewhat cleaner die surface than our manual method, but these machines use a lot of acid and need to be cleaned after use. So even professional chip analysis laboratories extract the chip manually if only a few packages have to be opened.

Most chips have a passivation layer of silicon nitride or oxide, which protects them from environmental influences and ion migration. This is not affected by nitric acid; chip testers typically remove it using dry etching with hydrogen fluoride, a process that is not as easily performed by amateur hackers.

But dry etching is not the only option. Another approach is to use microprobing needles that remove the passivation just below the probe contact point using ultrasonic vibration. Laser cutter microscopes commonly used in cellular biology laboratories have also been used to remove the passivation locally. Some testing laboratories have sets of nine microprobes so that the card bus can be read out during real time operation [12].

It is also normal to remove the passivation before using an electron beam tester to access on-chip signals, because the secondary electrons emitted by the chip surface accumulate a positive charge on the passivation layer which causes the signals to disappear after a few seconds. One might therefore think that such attacks would require dry etching facilities. However, in some experiments with an electron beam tester, we have found that the charge accumulation effect is less serious when the chip is still covered with a thin dirt layer of  $\text{HNO}_3$  and resin remains, which is probably weakly conductive. We suggest that a suitable weakly conductive layer might be deposited on top of the passivation layer as an alternative way of preventing the charge build-up.

### 2.3 Advanced attack techniques

The techniques described above have been successfully used by class I attackers — amateur pay-TV hackers, students and others with limited resources. We will now briefly describe some of the techniques available in professionally equipped semiconductor laboratories, of which there are several hundred worldwide. Some of these are situated in universities (three in the UK, for example), and it has happened that class I attackers get access to professional equipment in the course of student projects.

A recent article [11] gives an overview of a technique developed for reverse engineering chips at the Cavendish Laboratory in Cambridge. The authors of that paper first developed techniques for cleanly etching away a layer of a chip at a time. One innovation is a technique to show up N and P doped layers using the Schottky effect: a thin film of a metal such as gold or palladium is deposited on the chip creating a diode which can be seen with an electron beam. Images of successive layers of a chip are then fed into a PC with image processing system software that reduces the initially fuzzy image to a

clean polygon representation and identifies common chip features.

The system has been tested by reverse engineering the Intel 80386 and a number of other devices. The 80386 took two weeks, and it usually takes about six instances of a chip to get it right. The output can take the form of a mask diagram, a circuit diagram or even a list of the library cells from which the chip was constructed.

Once the layout and function of the chip are known, there is an extremely powerful technique developed by IBM for observing it in operation, without even having to remove the passivation layer. The tester places a crystal of lithium niobate over the feature whose voltage is to be monitored. The refractive index of this substance varies with the applied electric field, and the potential of the underlying silicon can be read out using an ultraviolet laser beam passed through the crystal at grazing incidence. The sensitivity of this technique is such that a 5 V signal of up to 25 MHz can be read [30], and we understand that it is a standard way for well funded laboratories to recover crypto keys from chips of known layout. When attacking a smartcard, for example, we would read the EEPROM output amplifiers.

The response of the protection community to attacks of this kind has been to develop ‘conformal glues’, chip coatings that are not merely opaque and conductive but which also strongly resist attempts to remove them, usually damaging the underlying silicon in the process. These coatings are referred to in a FIPS standard [17] and are widely used by the U.S. military, but are not generally available.

In addition to chip coatings, silicon features may be used to obscure the design. We have heard of design elements that look like a transistor, but are in reality only a connection between gate and source; and 3-input NORs which function only as 2-input NORs. Such copy traps may use holes in isolating layers or tricks done in the diffusion layer with ion implantation. However, the layer etching and Schotky techniques described above can detect them.

Another possibility is to introduce complexity into the chip layout and to use nonstandard cell libraries. However the chip still has to work, which limits the complexity; and nonstandard cells can be reconstructed at the gate level and incorporated in the recognition software.

A more systematic approach was employed in the U.S. government’s Clipper chip. This had a fusible link system in which the links that created a classified encryption algorithm and a long term device

key from an unclassified mask were fused after fabrication, and were made of amorphous silicon to make microscopy more difficult. In addition to this, the surface of the chip was ‘salted’ with oscillators to make electromagnetic sensor attacks more complicated.

Details of the fusible link technology can be found in a paper in the relevant data book [18], and from the scanning electron micrographs there, it is clear that — given enough effort — the secret information can be recovered by sectioning the chip (this technique has been used by the Cambridge team on obscure features in other chips). We are reliably informed that at least one U.S. chipmaker reverse engineered the Clipper chip shortly after its launch. However the attacks that discredited the Clipper chip used protocol failures rather than physical penetration [10] — a topic to which we will return later.

Sectioning is not the only way to reverse engineer a chip whose surface is well protected. For example, a recently declassified technique invented at Sandia National Laboratories involves looking through the chip from the rear with an infra-red laser using a wavelength at which the silicon substrate is transparent. The photocurrents thus created allow probing the device’s operation and identification of logic states of individual transistors [2].

The use of sectioning leads us to a more general, and relatively unexplored, topic — attacks that involve actively modifying the target chip rather than merely observing it passively. It is well known that active opponents can mount much more severe attacks on both cryptographic protocols and algorithms than passive opponents can, and the same turns out to be true when reverse engineering chips.

We understand, for example, that production attacks carried out by some pay-TV pirates involve the use of a focussed ion beam (FIB) workstation. This device can cut tracks in a chip’s metallisation layer, and deposit new tracks or isolation layers. It can also implant ions to change the doping of an area of silicon, and it can even build vias to conductive structures in the lowest layers of the chip. These machines cost several million U.S. dollars, but low-budget attackers can rent time on them from various semiconductor companies.

Armed with such a tool, attacks on smartcards become much simpler and more powerful. A typical attack involves disconnecting almost all of the CPU from the bus, leaving only the EEPROM and a CPU component that can generate read accesses. For example, the program counter may be left connected in such a way that the memory locations will

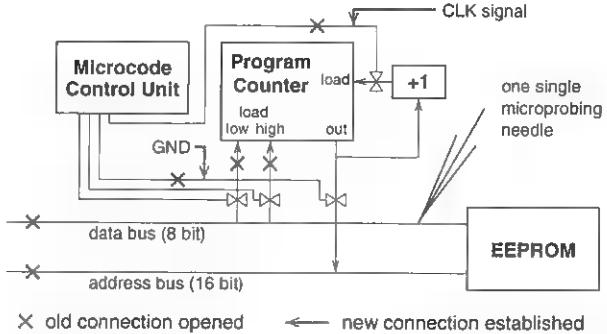


Figure 2: Read-out attack modifications on a security processor performed with a focused ion beam workstation allow easy access to secret EEPROM content with a single microprobing needle.

be accessed in order as the device is clocked (see Fig. 2). Once this has been done, the attacker needs only a single microprobing needle or electro-optical probe to read the entire EEPROM contents. This makes the program much easier to analyse than in passive attacks, which typically yield only an execution trace; it also avoids the considerable mechanical difficulties of keeping several probes simultaneously located on bus lines that are perhaps a micrometre wide.

In conclusion, it is imprudent to assume that the design of silicon chips, or the information stored in them, can be kept from a capable motivated opponent. So how can we protect key material from such an opponent?

#### 2.4 Advanced protection techniques

One application in which capable motivated opponents may be assumed, and where billions of dollars are spent on thwarting them, is the security of nuclear weapons. The threat model here is unequivocally class III — rogue states fronted by “terrorist” commando teams operating in cahoots with subverted military personnel. The U.S.A. has led the development of a control technology, now partially shared with other nuclear and near-nuclear nations, and the following account has been pieced together from a number of open sources.

Following the Cuban missile crisis, there was concern that a world war could start by accident — for example, by a local commander under pressure feeling that ‘if only they knew in Washington how bad things were here, they would let us use the bomb’. There was also concern that U.S. nuclear weapons in allied countries might be seized by the ally in time of tension, as U.S. forces there had only token custodial control. These worries were confirmed by three

emergency studies carried out by Jerome Wiesner, the presidential science adviser.

President Kennedy’s response was National Security Action Memo no. 160, which ordered that America’s 7,000 nuclear weapons then in countries from Turkey to Germany should be got under positive control, or got out [25].

The U.S. Department of Energy was already working on safety devices for nuclear weapons, the basic principle being that a unique aspect of the environment had to be sensed before the weapon would arm. For example, missile warheads and some free-fall bombs expected to experience zero gravity, while artillery shells expected to experience an acceleration of 20,000 g. There was one exception though: atomic demolition munitions. These are designed to be taken from their storage depots to their targets by jeep or helicopter, or even hand carried by special forces, and then detonated using time fuses. So there is no scope for a unique environmental sensor to prevent accidental detonation.

The solution then under development was a secret arming code, which activated a solenoid safe lock buried deep in the plutonium pit at the heart of the weapon. The main engineering problem was that when the lock was exposed, for example by a maintenance engineer replacing the power supply, the code might become known (as with the VISA security module mentioned above). So it was not acceptable to have the same code in every weapon, and group codes had to be used; the same firing code would be shared by only a small batch of warheads.

But, following the Kennedy memo, it was proposed that all nuclear bombs should be protected using code locks, and that there should be a ‘universal unlock’ action message that only the president or his legal successors could give. How could this be securely translated to a large number of individual firing codes, each of which would enable a small batch of weapons? The problem became worse when the Carter administration’s policy of ‘measured response’ created a need for a wide variety of ‘selective unlock’ messages, giving the president options such as enabling the use of nuclear artillery and air defence weapons against a Soviet incursion into Germany. It became worse still with concern that a Soviet decapitation strike against the U.S. national command authority might leave the arsenal intact but useless. As is now well known, the solution lies in the branch of cryptomathematics known as ‘secret sharing’ [24], whose development it helped to inspire, and which enables weapons, commanders and options to be linked together with a complexity limited only by the available bandwidth.

In modern weapons the solenoid safe locks have been superseded by PALs — prescribed action links — about whose design details we have been able to find no useful open source material. However, it is known that PALs are considered sufficient only when they can be buried in the core of a large and complex weapon. With simple weapons (such as atomic demolition munitions) it is not considered feasible to deny access to a capable motivated opponent. These weapons are therefore stored in sensing containers called PAPS (prescribed action protective system) which provide an extra layer of protection.

Both the big-bomb and PAPS-enhanced systems include penalty mechanisms to deny a successful thief access to a usable weapon. These mechanisms vary from one weapon type to another but include gas bottles to deform the pit and hydride the plutonium in it, shaped charges to destroy components such as neutron generators and the tritium boost, and asymmetric detonation that results in plutonium dispersal rather than yield. Whatever the combination of mechanisms used in a given design, it is always a priority to destroy the code in the switch; it is assumed that a renegade government prepared to deploy “terrorists” to steal a shipment of bombs would be prepared to sacrifice some of the bombs (and some technical personnel) to obtain a single serviceable weapon.

To perform authorised maintenance, the tamper protection must be disabled, and this requires a separate unlock code. The devices that hold the various unlock codes — for servicing and firing — are themselves protected in similar ways to the weapons. We understand, for example, that after tests showed that 1 mm chip fragments survived the protective detonation of a control device carried aboard airborne command posts, the software was rewritten so that all key material was stored as two separate components, which were kept at addresses more than 1 mm apart on the chip surface.

This highlights the level of care that must be taken when developing security processors that are to withstand capable attack. This care must extend to the details of implementation and operation. The weapons testing process includes not just independent verification and validation, but hostile ‘black hat’ penetration attempts by competing laboratories or agencies. Even then, all practical measures are taken to prevent access by possible opponents. The devices (both munition and control) are defended in depth by armed forces; there are frequent zero-notice challenge inspections; and staff may be made to resist the relevant examinations at any time of the day or night.

These mechanisms and procedures have so far succeeded in preventing rogue governments from stealing (as opposed to making) atomic weapons.

The nuclear business also supplies the only examples known to us of tamper resistant packages designed to withstand a class III opponent who can obtain unsupervised physical access. These are the missile sensors developed to verify the SALT II treaty [26] — which was never deployed — and the seismic sensor package developed for test ban treaty verification, which was. In this latter system, the seismic sensors are fitted in a steel tube and inserted into a drill hole that is backfilled with concrete. The whole assembly is so solid that the seismometers themselves can be relied upon to detect tampering events with a fairly high probability. This physical protection is reinforced by random challenge inspections.

So if systems have to be protected against class III opponents, we might hazard the following summary:

- if our goal is to merely detect tampering with a positive probability (as with treaty verification), then we can allow unsupervised access provided we are allowed to use a massive construction and to perform challenge inspections;
- if we wish to prevent the loss of a cryptographic key with near certainty (as with firing codes), then we had better use explosives and we had better also guard the device.

The above analysis convinced us that military agencies have limited confidence in the ability of tamper-resistant devices (and especially portable ones) to withstand a class III opponent with unsupervised access. Having read an early draft of this paper, a senior agency official confirmed that chip contents cannot be kept from a capable motivated opponent; at most one can impose cost and delay. A similar opinion was ventured by a senior scientist at a leading chip maker.

Furthermore, the expense and inconvenience of the kind of protection used in the nuclear industry are orders of magnitude greater than even major banks would be prepared to tolerate. So what is the state of the art in commercial security processor design? They may be vulnerable to a class III opponent, but how about class II and class I?

### 3 Commercial security processors

Many commercial systems use either security module or smartcard technology. However, a growing number of designs consist of a composite package

containing processor, memory, tamper detection circuitry and a battery.

An early example, whose design rationale was published in detail, is the  $\mu$ ABYSS coprocessor developed by IBM. A variety of tamper resistant packages were tested for ease of penetration and ease of manufacturing, including stannic oxide lines on glass, piezo-electric sheets and a number of wire winding techniques. The designers settled on a four layer wrapping of 40 gauge ( $80\ \mu\text{m}$  diameter) nichrome wire surrounding the processor, battery, memory and sensor circuitry, and embedded in a hard, opaque epoxy filled with silica to make it harder to machine and more likely to crack under UV laser ablation [28] [29].

This appears to be a promising technology, and increasingly so as circuit sizes and power consumption shrink. The  $\mu$ ABYSS design protected  $260\ \text{cm}^2$  of card circuitry, but much less is used in many recent designs. 128 kilobyte SRAM chips are available today with room temperature data retention currents of less than  $1\ \mu\text{A}$ . A small 3 V lithium cell can easily provide this for a decade.

Many aggressive chemicals used to remove opaque chip packages (such as fuming nitric acid) have a low electrical resistance and can easily be detected as long as battery power is available; indeed, they will often cause critical breaks and short-circuits directly. Power supply networks could be made from a variety of different conductive and isolating materials such that practically any useful chemical solvent will cause at least one part to fail.

Suitable packaging can make it difficult for the attacker to strip away the protection one layer at a time, so that a successful attack might require a highly laborious process of manually shorting out the protective wire winding, guided by X-rays and precise measurements of the voltage at various points along its length.

There are some subtleties though. One might think that the protection mechanisms only have to deactivate the power supply; but low-power SRAM chips remember bit values without a supply voltage at room temperature reliably for many seconds. By cooling the whole circuit with liquid nitrogen or helium, an attacker can extend the reliable power-off memory time to minutes or even hours, which could be enough to disable the alarm system and reapply power. Longterm exposure to a constant bit pattern can cause some SRAM cells to adapt their preferred power-up state accordingly, an effect that can remain for several days without any supply voltage

[19]. Possible countermeasures include SRAM cells with a well-defined power-up behavior.

Recent examples of battery-backed security module assemblies are the IBM Transaction Security System [1] and the Dallas Semiconductor DS5000 series [16]. The latter devices have been described by a European signals security agency as the most secure processors available on general sale; we will now report a successful attack on them.

### 3.1 The Dallas DS5002FP Secure Microcontroller

One might want to make the tamper resistant module as small as possible, since hermetic sealing limits power dissipation, because larger packages are more vulnerable to false alarms, and for simple cost reasons.

But many applications require much more RAM than can be conveniently included in a small package, and one established technique is bus encryption [13] [14] [15]. The CPU contains hardware for on-the-fly encryption of both the external address and the data bus. External RAM contains only encrypted data stored at encrypted addresses. The secret key is stored in a special battery buffered register on the CPU chip.

The Dallas Semiconductor DS5002FP microcontroller uses this bus encryption strategy. This Intel 8051 compatible processor is used in a number of financial transaction terminals and pay-TV access control systems to store secret keys and execute confidential cryptographic algorithms. On-chip bootloader firmware allows users to upload unencrypted software; it is then encrypted and stored in the external memory. The secret key is unique to each device, which has a special self-destruct pin that allows external alarms to erase it. A special version (DS5002FPM) features an additional metal layer die top coating designed to prevent microprobe attacks.

According to the manual, this layer is a “*complex layout that is interwoven with power and ground which are in turn connected to logic for the Encryption Key and Security Logic. As a result, any attempt to remove the layer or probe through it will result in the erasure of the security lock and/or the loss of encryption key bits*”. Additional security is provided by pseudo-random dummy accesses performed on the external bus whenever the CPU core does not access external memory. In addition, 48 bytes including the reset and interrupt vectors are located on chip. Access to them also results in external dummy RAM access cycles, such that anyone observing the external bus cannot know when the internal RAM is accessed.

The security features of the DS5002FP are at first glance quite impressive and the manufacturer describes them as “*the most sophisticated security features available in any microcontroller*”.

The chip uses two block ciphers that are loosely modelled on DES. The first encrypts addresses and acts on 15-bit blocks; the second encrypts data and acts on 8-bit blocks. The key of the second cipher is salted with the address of the byte being encrypted, but its small block size (which was no doubt dictated by the fact that the controller is byte oriented) turns out to be a useful feature for the attacker.

On closer examination, the algorithms show statistical weaknesses that might allow key recovery using differential cryptanalysis. We have not studied this in detail yet. In any case the algorithm strength is a purely economic issue; more rounds can buy more strength, but at a cost in either clock frequency or transistor count. Much more interesting is a weakness of the bus encryption system that is independent of the quality of the encryption algorithms.

### 3.2 Breaking the Dallas chip

One of us (Kuhn) has designed and demonstrated an effective practical attack that has already yielded all the secrets of some DS5002FP based systems used for pay-TV access control and also broken a code lock provided as a challenge by the German Federal Agency for Information Technology Security (BSI). The attack requires only a normal personal computer, a special read-out circuit built from standard electronic components for less than US\$100, and a logic analyzer test clip for around US\$200. It was performed in a student hardware laboratory at the University of Erlangen-Nürnberg using only common laboratory tools. Designing the hardware and software and performing the experiments leading to the final approach required less than three months. Thus, in the IBM taxonomy, this attack was carried out by a class I opponent.

The basic idea is simple, but was clearly not considered by the designers or evaluators of this processor. We call it the “cipher instruction search attack”: it works by feeding the CPU with suitably chosen enciphered instructions and looking to see which of the resulting plaintext instructions we can recognise from their effects.

For example, we can recognise the three byte instruction

MOV 90h, #42h

encoded 75h 90h 42h, as it outputs byte value 42h on parallel port P1 (address 90h) two bus access cycles later.

So we reset the CPU and wait until some target instruction is about to be fetched. Then our read-out hardware replaces it, and our control software observes the reaction for a few more clock cycles. Then we repeat the procedure — which we can do over 300 times per second — and systematically work through all  $2^{16}$  combinations for the first two encrypted instruction bytes.

We eventually find a two byte combination that sends a bijective function of the following byte to the parallel port. Assuming the first two bytes are the ciphertext corresponding to 75h 90h (which has to be confirmed by further tests), this gives the data bus decryption function at the address from which the third instruction byte was fetched. By testing all  $2^8$  values for this byte, we can tabulate the data decryption for one address.

Now we repeat the whole process. However this time we will search for a one-byte no-operation command (NOP) followed by the same MOV instruction as before. This effectively increases by one the address from which the third MOV instruction byte will be fetched.

Although we are now searching for a combination of four encrypted bytes representing two machine instructions, the search complexity has not been increased. We know already the correct encrypted value for one byte (port address 90h) from the previous tabulation of the encryption function at this address. The first instruction does not have to be a NOP, as any one-byte instruction that does not affect the following MOV will do. So the second search loop requires considerably less than  $2^{16}$  iterations — in fact we typically need less than 2,500 attempts.

This search process becomes steadily faster as more addresses are tabulated, and we quickly tabulate the encryption function for a number of consecutive but still unknown addresses. We are now able to encrypt and send to the processor a sequence of machine instructions that simply dumps all the memory and special registers to one of the I/O ports.

The attack is in reality somewhat more complicated than presented in this brief overview. The details will be presented in a separate publication, together with a discussion of possible countermeasures for future bus encryption based systems. Our point is that a class I attacker could circumvent the physical protection of the ‘top’ commercial system

with modest effort. As the attack did not exploit either physical or cryptographic weaknesses, it might be considered a kind of protocol attack [8].

## 4 Conclusion

It is prudent engineering practice to avoid single points of failure, and especially so where the likelihood of failure is unknown. This makes it all the more remarkable that the tamper resistance claims made for smartcards and other commercial security processors have gone untested for so long. The reader will by now be convinced that these claims should be treated with circumspection.

Public key techniques offer some solace, as the number of universal secrets can be greatly reduced — ideally, to a small number of certification keys, that can then be protected in depth. However, public key protocols have their own problems [9], and we should never forget that the great majority of actual security failures result from simple blunders in design, construction and operation [6] [7]. There is no silver bullet.

A prudent engineer will see to it that the penetration of a moderate number of accessible devices, such as smartcards or payment terminals, will not be disastrous for the system. As most current electronic wallet systems use symmetric cryptography with universal secrets stored in retailers' terminals, they should be designed to keep on working after these secrets have been compromised — such as by supporting a fallback processing mode similar to that for credit cards, with full reconciliation, intrusion detection, hot lists and security recovery.

But although it is necessary to design commercial security systems with much more care, it is not sufficient. They must also be subjected to hostile testing. Readers may compare the nuclear weapons community's insistence on independent verification and validation with the aversion of the banking industry to any hostile review of their systems [5]. It is encouraging to note that some other sectors, such as the prepayment electricity meter industry, are starting to recognise the value of hostile review [7]. Other industries, such as pay-TV, got their hostile review once their systems were fielded.

**Acknowledgements:** This paper was largely written while the authors were guests at the Isaac Newton Institute, Cambridge. We also acknowledge useful comments from Bob Morris, Gus Simmons, Louis Guillou and a number of sources who prefer to remain anonymous; and support from the Fondazione Ugo Bordoni.

## References

- [1] DG Abraham, GM Dolan, GP Double, JV Stevens, "Transaction Security System", in *IBM Systems Journal* v 30 no 2 (1991) pp 206–229
- [2] C Ajluni, "Two New Imaging Techniques Promise To Improve IC Defect Identification", in *Electronic Design* v 43 no 14 (10 July 1995) pp 37–38
- [3] A Anderson <[aha@apollo.HP.COM](mailto:aha@apollo.HP.COM)>, message posted to USENET *sci.crypt* 26 Apr 1994, message-ID <[CovCG9.581@apollo.hp.com](mailto:CovCG9.581@apollo.hp.com)>
- [4] RJ Anderson, "Crypto in Europe — Markets, Law and Policy" in *Cryptography: Policy and Algorithms*, Springer LNCS v 1029 pp 75–89
- [5] RJ Anderson, "Liability and Computer Security: Nine Principles", in *Computer Security — ESORICS 94*, Springer LNCS v 875 pp 231–245
- [6] RJ Anderson, "Why Cryptosystems Fail", in *Communications of the ACM* v 37 no 11 (Nov 94) pp 32–40
- [7] RJ Anderson, SJ Bezuidenhoudt, "On the Reliability of Electronic Payment Systems", in *IEEE Transactions on Software Engineering* v 22 no 5 (May 96) pp 294–301
- [8] RJ Anderson, RM Needham, "Programming Satan's Computer", in *Computer Science Today*, Springer LNCS v 1000 pp 426–441
- [9] RJ Anderson, RM Needham, "Robustness Principles for Public Key Protocols", in *Advances in Cryptology — CRYPTO 95*, Springer LNCS v 963 pp 236–247
- [10] M Blaze, "Protocol Failure in the Escrowed Encryption Standard", in *Proceedings of the 2nd ACM Conference on Computer and Communications Security* (2–4 November 1994), ACM Press, pp 59–67
- [11] S Blythe, B Fraboni, S Lall, H Ahmed, U de Riu, "Layout Reconstruction of Complex Silicon Chips", in *IEEE Journal of Solid-State Circuits* v 28 no 2 (Feb 93) pp 138–145
- [12] E Bovenlander, RL van Renesse, "Smartcards and Biometrics: An Overview", in *Computer Fraud and Security Bulletin* (Dec 95) pp 8–12
- [13] RM Best, "Microprocessor for Executing Enciphered Programs", U.S. Patent No. 4,168,396, September 18, 1979

- [14] RM Best, "Preventing Software Piracy with Crypto-Microprocessors", in *Proceedings of IEEE Spring COMPCON 80*, pp 466–469
- [15] RM Best, "Crypto Microprocessor for Executing Enciphered Programs", U.S. Patent No. 4,278,837, July 14, 1981.
- [16] 'Soft Microcontroller Data Book', Dallas Semiconductor, Dallas, Texas, 1993
- [17] 'Security Requirements for Cryptographic Modules', FIPS PUB 140-1, Federal Information Processing Standards Publication, National Institute of Standards and Technology, U.S. Department of Commerce, January 11, 1994
- [18] KE Gordon, RJ Wong, "Conducting Filament of the Programmed Metal Electrode Amorphous Silicon Antifuse", in *Proceedings of International Electron Devices Meeting*, Dec 93; reprinted as pp 6-3 to 6-10, *QuickLogic Data Book* (1994)
- [19] P Gutmann, "Secure Deletion of Data from Magnetic and Solid-State Memory", in *Sixth USENIX Security Symposium Proceedings*, San Jose, California, July 22–25, 1996, pp 77–89
- [20] D Kahn, 'The Codebreakers' (Macmillan 1967)
- [21] P Maes, marketing director, Gemplus, *comment during a panel discussion at Cardis 94*
- [22] R Morris, *talk given to Cambridge Protocols Workshop*, 1994
- [23] W Rankl, W Effing, 'Handbuch der Chipkarten', Carl Hanser Verlag, 1995; ISBN 3-446-17993-3
- [24] B Schneier, 'Applied Cryptography – Protocols, Algorithms, and Source Code in C' (second edition), John Wiley & Sons, New York, 1996
- [25] GJ Simmons, invited talk at the *1993 ACM Conference on Computer and Communications Security*, Fairfax, Virginia, Nov 3–5, 1993
- [26] GJ Simmons, "Subliminal Channels; Past and Present", *European Transactions on Telecommunications* v 5 no 4 (Jul/Aug 94) pp 459–473
- [27] 'VISA Security Module Operations Manual', VISA, 1986
- [28] SR White, L Comerford, "ABYSS: A Trusted Architecture for Software Protection", in *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press pp 38–51
- [29] SH Weingart, "Physical Security for the μABYSS System", in *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, pp 52–58
- [30] JM Wiesenfeld, "Electro-optic sampling of high-speed devices and integrated circuits", in *IBM Journal of Research and Development* v 34 no 2/3 (Mar/May 1990) pp 141–161; *see also subsequent articles in the same issue*



# Token-Mediated Certification and Electronic Commerce

Daniel E. Geer, Jr., Sc.D., Donald T. Davis

*Open Market, Inc.*

*Cambridge, Mass. 02142*

*{geer, ddavis}@openmarket.com*

## Abstract

Public key technology presumes the availability of certificates and certifying authorities (CAs) living within a shallow hierarchy rooted at a few ( $n \ll 100$ ) public CAs. We propose an alternative that lessens the day-to-day dependence on centralized CAs while deepening the certificate tree. We do this by suggesting that smartcards provide CA functions, thus re-framing some payment problems as simpler authorization problems.

## 1. Rationale by way of introduction

Smartcards will, beyond any doubt, dominate all commerce and will be how electronic commerce and conventional commerce merge into a single commerce. Irresistible global forces are behind this, and no one company has either the mass or the momentum to change it. Getting it right will have historical significance. We think we have — we encapsulate transactional commerce in public key certificates issued by token devices. These ubiquitous, personal certification authorities would not supplant centralized CAs, but would perform temporary local name-to-key bindings, primarily for access-control. We suggest that by easing some of access-control's scaling problems in this way, we gain the opportunity to re-frame some payment problems as simpler authorization problems.

The peculiar advantage of smartcard-issued certificates is that they are temporary and cheap because they are not public documents. A smartcard-issued certificate is the public key equivalent of a symmetric session key; we've brought together the benefits of asymmetric key-management and of low-cost, low-value temporary keys. Indeed, smartcard-issued certificates do not require the same kind of trust that conventional public key certificates do. Instead, a personal CA's certificates are in a sense reflexive: when Bob's smartcard issues a certificate for Alice, Alice will use the cert not to communicate with others, but only for her future communications with Bob. A personal CA could operate in this way without a smartcard, but installing this function in a smartcard protects the

bearer from theft of the CA's private-key.<sup>1</sup>

## 2. One, two, many CAs

We suggest a novel idea: suppose a smartcard had the ability to issue public key certificates? How might such CA-smartcards be advantageous in electronic commerce? But before we list what smartcard-CAs can offer for merchants and consumers, we'll describe a few transaction scenarios. The reader will note that in each example the smartcard-issued cert is a single-use delegated authorization granting access to the issuer's resource.

- A consumer uses his smartcard to buy a video-download from Sony's Web page while he's at work. The smartcard issues an authorization certificate to Sony's download server. Sony's server calls the consumer's home-PC's modem, and presents the authorization certificate in a "May I download?" request. The consumer's home-PC uses the consumer's *public-key* to validate the cert, and permits the download. By the time the customer gets home, the video is ready to watch.
- A consumer uses his smartcard to subscribe to an online magazine, but the magazine doesn't appear on a Web site (because the Web is slow), and it doesn't appear in his mailbox (because the magazine is too bulky for e-mail). Rather, the

<sup>1</sup> For the purposes of this paper, we use the term "smartcard" to mean a handheld cryptographic token that communicates via a reader with the desktop CPU, and whose functions are not limited to memory insertion and retrieval. We include neither one-time password tokens nor memory only "chip cards" in this class.

magazine spontaneously appears on the consumer's local disk, or in his home-directory. The consumer does receive a brief e-mail notice of each magazine's arrival. The consumer's computer accepts the magazine's asynchronous download, because the publisher presents an authorization certificate that the consumer's smartcard issued when the consumer subscribed.

- A purchasing officer needs to buy something that requires countersignature. He generates a new public key pair and sends a combination of his certificate and the (sealed) new public-key to a higher-ranking officer. The higher-ranking officer mints a new certificate, one that has both the increased authority and a limitation thereon for the particular purpose at hand. Possibly, the cert is returned encrypted in the public-key of the card so that it is only usable after a session in which the card is involved.

Alternately, the senior officer can craft the public key pair on behalf of the junior officer. The relevant private-key is returned within the cert either singly sealed in the (junior officer's) role's public-key or doubly sealed in the role's public-key *and* the public-key of the junior officer's smartcard's public-key (thereby denying the opportunity for further delegation). Such a scenario, the senior officer generates the public key pair, is inherently consistent with escrow. In either case, the junior officer can use that certificate as an otherwise ordinary client certificate in a transaction to which he is not otherwise entitled.

- An executive wishes to delegate to a subordinate the right to process the executive's e-mail while the executive is on a 10 day vacation. The executive issues a cert for the role "process executive's mail" with limited time-validity. Because the cert is a delegation cert, the cryptographically signed mail will be signed with the delegatee key, *i.e.*, "on behalf of."
- An investor issues a cert to a broker combining a "limit order" and a "trade at close" directive, *i.e.*, an order to trade at a range of prices if those prices obtain near a market's closing moment. Such a cert satisfies all the ordinary requirements of a secure communication, *viz.*, it is verifiably authentic, authoritative, non-repudiable, and can be issued such that it must be confirmed at the moment of use (permitting revocation thereby). Members of the exchange could issue certs that enable investors to book trade orders directly through to the floor of the

exchange with such certs containing whatever volume or credit limits as represent the assumed risk of trade delegation to the investor.

- A consumer writes a cert instead of a paper check.<sup>2</sup> The cert authorizes the consumer's bank to debit the consumer's checking account upon presentation by a named or un-named identity. A "stop payment" is a kind of certificate revocation, and remains a single message between the consumer and his bank. It, too, has a transactional semantic.
- The consumer uses a "personal-proxy webserver" such as Open Market's OM-Express to download a catalog, does off-line shopping, and places the order whenever it is convenient to do so by uploading a collection of "purchase" certs.

In each of these scenarios, the CA-smartcard (CA-sc) performs a handshake that authenticates both parties and converts a new public-key into a new certificate. We will rely on X.509v3 extensibility by including, as an attribute-value pair, the ancestor certificate that contributes to the current one. In this way, it will not be possible to disguise (or repudiate) the delegation of authority that the cert chain represents. The flow would be:

CA-sc <--> merchant	exchange public key certs
CA-sc <--> merchant	mutual authentication, session-key
merchant --> CA-sc	new public-key, MAC
CA-sc --> merchant	new cert: "Merchant may download" <i>time may pass</i>
merchant --> consumer	new cert, req-to-download
consumer --> merchant	consumer-cert, mutual auth, "OK"
merchant --> consumer	purchased data

As usual, the CA (smartcard CA or CA-sc here) never sees the other party's private-key. The new certificate is essentially a proxy authorization, but with a reflexive flavor; it is a grant of authority by me to myself and on my own behalf, rather than a letter of introduction to others yet un-named. This reflexivity avoids questions of trust management [BlFeLa96] because the user is simply trusting his own key to represent his past wishes to him himself. The consumer authorizes the merchant to manipulate the consumer's resources or data. The merchant exercises this delegated right, on the consumer's behalf.

---

<sup>2</sup> Or, a consumer writes a cert instead of an electronic check such as in the NetCheque system [NeMe95] which quite cleverly preserves the flow and semantics of paper checks in an asynchronous electronic medium such as e-mail.

CA-smartcards make several things easier in electronic transactions:

- #1 Fulfillment can be greatly delayed after payment, without access-control administrivia on either end. In particular, no ACL is necessary (a huge scalability bonus).
- #2 The merchant can deliver the service or product asynchronously. Current Web-commerce assumes that the customer initiates the fulfillment connection.
- #3 The consumer can receive services and products without interactivity, without conscious effort, and without using his private-key.
- #4 Delegation of roles becomes easy and assumes a transactional semantic that can include counting functions.

And why are these important?

- #1 Makes electronic commerce easier to set up and manage, for everyone.
- #2 Will make certain services easier to commercialize. For example, a long-haul bulk-transport service might accept a source-URL, a target URL, and an authorization cert from the customer (at payment-time). Later, when the service's proprietary lines become available for a gigabyte transfer, the service could use the cert to authenticate itself to the source and target, before starting the transfer. The customer need not be online during the transfer itself. In this area, new services are enabled rather simply automating traditional commerce.
- #3 Makes electronic shopping easier and more transparent for consumers. It also becomes safer, because the home computer can receive delivery even in the consumer's absence, without exposing the smartcard or the CA's private-key to theft.
- #4 In the end, we believe that delegation of roles and commercial transactions merge. (A businessman may not care about this merger. Yet.)

### 3. Smartcard CAs: Core of the idea

When Bob's smartcard issues a cert to Alice, no one else has any reason to trust Bob's claims about Alice's key. Alice's cert is really only a message from Bob to himself, saying in effect, "Earlier, Alice authenticated herself once with this key-pair and once with a trusted one, so I know that this key is hers."

Alice's cert validates her temporary public-key to Bob; this gives her a quick way to re-authenticate with Bob, or for him to send private messages to her, without Bob's having to remember a shared symmetric-key for long periods of time. Note that Alice and Bob could achieve the same effect with other mechanisms, but certification seems to be the simplest and most natural.

Further, with X.509 V3's extended certificate formats, Bob can include in Alice's cert her access-permissions for Bob's resources. Issuing such an authorization cert relieves Bob of the need to store or remember Alice's rights, and this approach, pursued adroitly, turns out to have scaling benefits. Finally, and crucially, Bob can receive, validate, and act upon this authorization cert, without using his smartcard or his private-key. This means that even in Bob's absence, Bob's computer can securely act on his behalf, without using his private-key and thus without exposing it to theft. This no-risk activity is unusual in public key cryptography.

## 4. Needs of electronic commerce

As the Internet prepares to deploy public key cryptography on a large scale for commercial applications, many in the industry eagerly welcome its unusual security features:

- Extreme cryptographic strength,
- Trustless administration,
- Anonymous cash protocols, and
- Non-repudiable signatures.

However, we argue that these features are not particularly necessary or useful for electronic commerce. We believe that the main commercial value of public key crypto lies in deeper, less-appreciated features:

- Asynchronous authentication,
- Geographic reach, and
- Access-control scaling.

Smartcard-issued certificates help public key to fulfill these deeper needs of electronic commerce. In the rest of this section, we will justify this reassessment of public key cryptography. First, we will discuss these true merits, and then we will explain our depreciation of public key's supposed merits.

### 4.1. Why Public Key Lends Itself to Commerce

## Fast start

Much of public key's advantage over symmetric key security is transitory, because public key authentication is fairly costly, except when no security infrastructure is available. Once a security infrastructure is in place for all of the Internet, symmetric key authentication will offer lower per-transaction costs than public key. Ordinarily, public key cryptography is deemed essential for electronic commerce because (where its infrastructure is in place) public key cryptography absolutely minimizes the marginal cost of set-up that must precede a secure transaction between two commerce-ready parties. Stating this again for emphasis: Public key technology minimizes the cost of set-up that must precede a secure communication; it does not necessarily minimize overall security costs nor does its cost structure adapt to the risk involved in a particular transaction, nor do the costs and benefits distribute evenly. Nevertheless, we defer further discussion of cost minimization to other fora and merely declare that public key technology will dominate commerce.

## Geographic Reach

Public key cryptography allows self-initiated secure communication between people who have no security administration in common. While this geographic reach will contribute greatly to the growth and adoption of electronic commerce, it probably won't be that important once commercial networking technology matures. Public key technology is particularly advantageous when key-bearing users are too thinly spread across the network to be united efficiently in a single security administration. That is precisely the current state of electronic commerce: relatively few consumers possess cryptographic credentials of any kind, and few of the institutions on the network are preparing to issue cryptographic credentials. Thus, if a buyer and seller on the Internet have cryptographic keys at all, their keys are unlikely to come from the same place. Further, when on-line security servers are rare, it's unlikely that a widely-separated buyer and seller will both have access to the same security server. In this situation, compatibility of disparate credentials is easier with public keys than it would be with symmetric key cryptography, and users are better off without

having to rely on an online intermediary.

However, as commerce becomes pervasive, buyers and sellers will be more likely to get their credentials from the same issuing authority, wide-area symmetric key-distribution-centers (KDCs) will be more pervasive, too, and public key will lose its geographic advantage. For example, if one or a few ISP's gain national hegemony, they may prefer to offer symmetric key credentials to clients, and to use public key credentials for inter-server applications. Thus, in essence, public key cryptography will provide a long-distance scaffold, on which we will build a national infrastructure for commerce.

## Asynchronous Authentication

Perhaps the most important feature of public key cryptography is that its basic mode of operation is asynchronous; it does not require two-way network connections. In fact, the sender and recipient of an authenticated message need never be on the network at the same time.<sup>3</sup>

Asynchronous messaging is well-suited to the complex multi-party protocols that will likely be common in electronic commerce. For example, a typical electronic transaction might involve six or more communicating parties, all widely separated on the network:

- a customer,
- a merchant or content provider,
- their 2 financial institutions,
- an authentication service such as a CA,
- an access-control manager.

Asynchronous messaging can allow the transaction to conclude with a fairly simple set of protocols, without maintaining several simultaneous connections amongst the participants, and without imposing extra processing-state on any of the central servers. Thus, correct use of public key cryptography can make the design of secure and failure-tolerant protocols easier than with symmetric key cryptography, albeit at some cost in speed and key-management. [Da96]

---

<sup>3</sup> Asynchronous messaging is possible with symmetric key systems too, but the public key mechanism is simpler. See [DaSw92] and [LaAbBuWo91].

## 4.2. Bad Reasons to Use Public Key for Commerce

### Perfect Strength:

Though public key offers the possibility of unbreakable security, this ideal requires such long keys as to obstruct real-time applications. In practice, commerce applications often choose short key-sizes so as to keep response-times and computational burdens reasonable. For example, VISA's STT payments protocol [VI90] specified 512-bit key-pairs for client certificates even though the insecurity of such short keys had widely been accepted since Lenstra's success in factoring RSA-120, fully two years earlier. [DeDoLeMa93]

Most RSA users and application developers are more conservative and opt for 768-bit or 1024-bit keys, on the grounds that these lengths give good security for an acceptable performance cost. But, the expert opinion is that these key-lengths too will fall within 10 and 20 years, respectively. [Od96] Thus, RSA's potential for extremely high security is limited by the practical need for fast protocols.

This area is changing quickly. The rise of elliptic curve algorithms may well make the RSA patent worthless before it expires, but elliptic curve crypto does not bring perfect security within reach. Elliptic curve does run much faster than the public key algorithms it replaces (Diffie-Hellman and DSA), but EC is not so fast as to outrun this performance/security tradeoff.

### Trustlessness

Public key protocols do not expose clients' keys to administrators, so even if a PK administrator is corrupt, he can't learn clients' secrets, and he can't impersonate clients. However, this property isn't important in commercial operations, which invest trust in many other people besides the security administrator, anyway. Trust relationships are a normal part of commercial and corporate existence. For example, corporations routinely entrust sensitive documents and plans to couriers, parcel-delivery services, advertising agencies, financial institutions, and corporate partners. Businesses also necessarily place a great deal of trust in individuals, including company staff, consultants, and lawyers. Similarly, citizens and consumers place comparable trust in corporations, in civil authorities, and in other people, during the course

of our everyday economic activities.

### Anonymity

Another form of commercial trust holds when people and corporations trust one another to keep some transactions private. Several proposed electronic cash mechanisms have been designed to allow consumers to make anonymous electronic purchases. However, in conventional commerce, explicitly anonymous transactions are rare, and guarantees of anonymity are rarer still. Cash is technically anonymous, but cash expenditures are really only quasi-anonymous because we usually use cash in face-to-face purchases. There are several reasons for anonymity's rarity in conventional commerce:

- Anonymous transactions are hard to arrange and protect,
- Civil authorities have found that anonymity can undermine tax laws, and
- Anonymity is most necessary in illicit commerce.

Electronic commerce can make anonymous transactions easier to perform, but can do little about the other constraints. Thus, we see little need in electronic commerce for anonymous network mechanisms.

### Non-Repudiation

Perhaps the most dramatic feature of public key cryptography is the non-repudiable digital signature, which does nothing to hide an electronic message's content but instead assures every reader of the message's authenticity. Digital signatures are almost perfectly analogous to handwritten document signatures, so it seems that electronic contracts could become commonplace in commerce, once the legislatures and courts agree that digital signatures should be legally binding.

It seems to us, though, that digital contracts are a solution in search of a problem. Crucially, how will digital contracts reduce transactional labor costs or improve a business' economies of scale? Without addressing cost-reduction needs such as these, digital contracts cannot attract markets and customers to the Internet.

Electronic commerce, growing as it is on an untrustworthy medium, tends toward a more paranoid trust model than a secure Internet would inspire. For

example, signatures and anonymity have loomed larger than necessary in electronic commerce discussions because the Internet's insecurity arouses an unfocused and all-encompassing distrust in all of us. Eventually, commerce networks will provide "good-enough" end-to-end security transparently, electronic commerce's social cynicism will subside, and electronic commerce will fall back on traditional commercial trust-models, by-and-large. People will remember that after all, non-reputable contracts aren't necessary for most transactions.

So, we disagree with the common claim that electronic commerce needs public key technology so as to gain absolute privacy, trustless administration, perfect anonymity, and electronic contracts. While we disparage these claims, we still accept the value of public key cryptography in commercial networking.

We believe that electronic commerce's primary benefit is scale, and that security issues are secondary. With networked marketing and networked payments, merchants gain larger markets and lower labor costs for transaction processing. Banks and credit-card companies expect to gain ■ high volume of per-transaction fees. Accordingly, to meet these expectations, public key's contribution to electronic commerce should address scaling issues. It is this need, at bottom, that smartcard-CAs address.

## 5. Raw Material

If we want to build a commerce system that includes smartcard-CAs, what do we have to work with? We suggest that it is smartcards, client software, certification, and delegation.

### 5.1. Smartcards

Treating this in biologic style, a "card" is a family of devices that "fit in ■ wallet." It has two genera, a "dumbcard" which is a passive device and a "smartcard" which is an active device. Within dumbcards, there are magnetic and non-magnetic species, *i.e.* requiring a reader or not requiring a reader. Within smartcards, there are wallet cards and PC/MCIA cards, distinguished mostly by the additional capacity that the much increased silicon real estate that PC/MCIA affords. All smartcards are gaining cost-effective capacity quickly, and some predict that they will shortly include biometrics such as thumbprint readers on wallet cards.

The availability of readers will sort itself out quickly. German Point-of-Sale conversion is happening now, Fischer International has a reader in a 3.5" floppy case, the Defense Logistics Agency began requiring PC/MCIA-capable machines beginning earlier this year, PC/MCIA-based smartcard readers are now circa \$100 and cell phones and network computers alike now have the implicit premise of a smartcard as their chief protection. Other such examples seem to be appearing weekly.

We are interested in both the credit-card and the PC/MCIA cards which have a cryptographic co-processor and the ability to store at least two public key certificates, one for the card itself and one for the holder of the card. (With respect to some commercial systems, notably VISA's SET and Nortel's Entrust system, separate key pairs for signing and for sealing are required, so the number "2" may be higher.) We suggest the minimum of two certificates and a cryptographic co-processor to permit secure key management on the card, to admit the role of a security officer, and to separate "memory-only" functions from smartcard functions.

For important transactions to absolutely rely upon the smartcard, the private-key-half (of the public key pair) must never leave the card (the "trusted computing base"), but corporate convenience may well lead to key pair generation and escrow offboard the card. Regardless, we assume that the browser will be able to retrieve the user's "client certificate" whether via ■ PlugIn, a JAVA applet, or via some native browser capacity.

### 5.2. Client software

Client software to use such cards will likewise sort itself out quickly. Netscape is actively looking to add both PlugIn and direct browser support for public key smartcards, and is rumored to have engaged an offshore company to incorporate smartcard support (and full strength crypto) directly into the Netscape Navigator Web browser. Microsoft is similarly inclined; Microsoft's Crypto API performs client-side authentication based on certificates, secure certificate storage, and basic encryption/signature services. This Crypto API supports smartcards, in that smartcards can plug into the Crypto API layer from below to export their services to application software.

### 5.3. Certification and certs

In this section, we review the basics of public key infrastructure: certificates and certification authorities.

In commerce as in other social matters, people use names to identify other people, singly and in groups. However, the Internet community has learned through hard experience that simple exchange of names doesn't work well in packet communications. Computers can identify one another reliably only by their use of encryption keys. As the purpose of a certificate is to publicly bind a name to a key, we transpose our social identities into this electronic medium by means of public key certificates. The certification authority's job is to perform this transposition faithfully.

An RSA key-pair is in itself an electronic identity. When we see two widely-separated messages, both signed with the same private-key, we know the messages come from the same author, even if we don't know the author's name. In other words, without a certificate to bind a name ("identity") to that key we nonetheless have "reproducibility;" for many electronic applications, this is all we need.

For example, a file server might reasonably grant administrative rights to any client that can sign messages with an admin key-pair  $P_a$ . The file-server has no interest in the administrator's social identity, and the access-control mechanism can perfectly consistently "name" administrators by their keys, instead of by their conventional UIDs. This idea of using public keys as names is a cornerstone of the Simple Public Key Infrastructure (SPKI) proposal. [El96]

For commerce, though, we need more than electronic consistency. A certificate is what tells us, "Only Alice owns key-pair K." From this, we can hold Alice's social identity responsible for her electronic activities, as when:

- A merchant must re-possess Alice's purchase for non-payment,
- Her Web store's sales-tax payments are past due, or
- Her "GET RICH NOW" pyramid scheme collapses.

More generally, there are many commercial transactions in which prepayment is impossible and, in all such cases, plaintiffs need recourse to street addresses, summonses, and hard cash. At bottom, this is why "named key" certificates are necessary: to bind our electronic identities to our social identities.

In conventional applications of public key cryptography, public key certificates are issued only by trusted services called "Certification Authorities." A CA signs messages of the form, "Alice's public-key is NNNNNNNN." The CA's job is to check Alice's identity before signing the cert, and perhaps also to ensure that she actually holds the corresponding private-key. Usually, the CA verifies Alice's identity by looking at her driver's license or some other photo ID. Other users can use NNNNNNNN to prove Alice's identity, but only after they verify the CA's signature on Alice's cert. The user community trusts the CA not to bind one person's name to another person's key.

Practical resistance to attack on public key based security requires that public keys be packaged as certificates, and, given these constraints, certifying authorities are essential to commerce. In fact, the service of a public certifying authority is the only part of commerce that neither merchants, buyers nor financial processors provide and which, in addition, does not lend itself to being a commodity.

Certifying authorities are difficult to operate, the business model is hardly yet worked out, revocation and roll-over remain unfunded liabilities, etc. Indeed, the risk-benefit equation of operating a commercial certifying authority has been such that even superbly well prepared, widely experienced players (such as BBN) have demurred. We should also note that commercial CAs (such as VeriSign) have tried without success to sell certs as a commodity.

In calling CA services a scarce commodity with substantial barriers to entry, we describe a problem we wish here to solve. We propose an alternative that lessens the day-to-day dependence on centralized CAs while deepening the certificate tree.

### 5.4. Commerce ■ Delegation

The term "commerce" is slippery, but we here define it to mean simply a transactional exchange between two parties. In particular, the participants are not required to live within the same organizational structure. In this sense commerce can be a small dollar-value retail purchase but it can also be counter-signing a personnel report. Practically, we focus on business settings though we do not believe this to be crucial.

Going one step further, if the reader can accept a claim that money is simply generic authority, then a purchase and a delegation are virtually identical in the

sense of ■ transfer of authority. It is a subtle but small step; colloquial use has heretofore associated "transaction" with the mutual exchange of value and "delegation" as but half that, a one-way transfer. Overall, this commercial use of SC-coined delegations has the advantage that it does commerce securely yet with fewer special-purpose financial protocols. We believe that as a primitive, SC-certs can simplify financial protocols generally. Our reasons for this claim are:

- Commercial transactions usually involve more than two or three parties,
- Adding secure connections to a protocol adds a lot of complexity,
- Some parties' involvement can be reduced to asynchronous authorization messages,
- Creating and interpreting authorization certs is straightforward.

Even though designing public key authentication protocols is harder than commonly appreciated [AnNe95], we believe that authorization security is easier than multi-party authentication.

For example, If Alice wants to buy ■ movie from Sony, she issues two certs to Sony: a "Sony may download" cert, and a "Sony may withdraw \$3 from my account" cert. Insofar as it's true that sc-certs can replace e-comm protocols, this may be because we're breaking down the synchronous connections of the current transaction models. Sony gets two certs from Alice and uses them later at disparate times. This breaks a protracted & secure application into three almost unrelated messages, so the result doesn't look much like a single protocol.

## 6. Authorization

There are three great styles of authorization, to wit, authentication of principals coupled to distributed ACLs, role-authentication coupled to policy directives either in software or policy databases, and capabilities with varying degrees of coupling to (an owner's) identity. We suggest the following bet, *viz.*, the authorization solution that matters in an electronic world is that of roles. The "role" we mean is an authenticatable identity but an authenticatable identity which is that of a permission class (group) rather than of an instance (individual).

Role based access control has ■ rich literature. It stands somewhere between discretionary and

mandatory access controls.<sup>4</sup> Since "authority" can be reduced simply to access to credentials, the expressive power of the X.509v3 certificate extensions can trivially accommodate roles (as v3 certs). Simply put, our bet is that roles will win.

- They will win because pure authentication technology can be reused for authorization, because roles lend themselves to the expression and enforcement of corporate and organizational policies, because roles reflect the structure of business organizations, and because they maximize security impact while minimizing costs, especially design costs.
- A capability is a token that in and of itself is sufficient to gain access to a resource, *e.g.*, ■ bearer bond. Capabilities are too hard to define and too hard to protect. It can be argued that money is the purest capability; it certainly is the most widely applicable form of authorization. Electronic capability systems have been built, including anonymous electronic cash mechanisms. [Ch92]
- Classic ACL solutions<sup>5</sup> — Central authentication with per-function ACLs synchronized across all servers offering the function are explicitly not scalable and will lose. ACL solutions that attempt to convert corporate policy into explicit enumeration of authorized principal identities suffer from both inconsistent conversion and important synchronization difficulties with respect to the changing pool of authenticable principals.

Putting it differently, roles win because:

Costs	Access-Control Mechanisms		
	ACLs	Capabilities	Roles
hard/easy to manage?	hard(-)	easy(+)	easy(+)
hard/easy to think?	easy(+)	hard(-)	easy(+)
hard/easy to steal?	hard(+)	easy(-)	hard(+)
Net Worth	+1	-1	+3

---

<sup>4</sup> Under discretionary access control the resource-owner's authority is sufficient to share information, whereas under mandatory access control information is shared only according to its own classification. Role-based access control strips the resource-owner of the authority to share information, *per se*, preferring rather to have the authorization authority be a separate security service, rather than the owner or the information itself.

<sup>5</sup>Clients provide authentication credentials to servers which look up clients in local permission databases.

## 7. Loose Ends

A further elaboration: For the time being, the easiest way for a consumer to receive purchased data might be to offer a secured FTP service. That is, the consumer sets up an FTP server that only accepts connections from clients bearing certs that the consumer's own smartcard has issued. Initially, this might be most useful for bulky entertainment media and for subscriptions to asynchronous news delivery.

It's also worthwhile for merchants to use smartcards to issue certificates. A merchant's CA-smartcard would issue authorization certs to customers. A customer's merchant-issued certificate constitutes proof-of-payment and, at the same time, the cert relieves the merchant of having to manage an ACL database. In essence, the circulating certs are the ACL database, in a distributed form. This use of CA-smartcards is similar to the current model of Web commerce, except that a merchant with a CA-smartcard doesn't have to manage an ACL, making it easier to allow (or require) a considerable time to pass between the customer's payment and the server's fulfillment. (Futures trading, anyone?)

One detail that we cannot neglect is revocation. Most CA-smartcards will issue short-lived certs, but, even so, revocation will sometimes be necessary. For example, if Alice issues a cert that authorizes a publisher to download stuff to her for a year and the publisher loses control of the certificate's private-key, she will need to be able to tell her machine to reject that certificate. It will not be necessary for her to manage CRLs, though; the merchant can check the cert it holds against the CRL service before each use and will receive a signed, time-stamped "cert OK" message. The merchant can send the "cert OK" message along with the cert itself at each use. Of course, it may be preferable to either issue a dozen certs at the outset, to re-issue the next cert with each cert used, or even to re-issue a counting cert (in an S-Key chaining style). Each of these represents introducing opportunities to revoke as well as moving in the direction of micro-certs.

For a system to be at once browser-independent and require no client-end software other than the browser, the natural solution is to use redirection of specially constructed URLs as a poor-man's RPC. Today we have

- Series of custom interactions between the browser and the merchant
- Redirect of the browser to a transaction manager

- Exchange of the browser's authentication credentials for authorization credentials issued by the transaction manager

- Redirect of the browser to the fulfillment site

- Exchange of both the browser's authentication and authorization credentials for synchronous delivery of the bought goods

Alternately, we propose:

- Series of custom interactions between the browser and the merchant

- Redirect of the browser to a transaction manager

- Exchange of the browser's authentication credentials for authorization credentials issued by the transaction manager

- Redirect of the browser to the fulfillment site

- Exchange of the browser's authorization credentials alone for asynchronous delivery of the bought goods

There are many advantages to simply using certs rather than vendor-specific structures, including adaptive reuse of the authentication technology that will inevitably be certificate-based. None of this is possible without smartcard (cryptographic co-processor) support, and smartcards plus this certificate strategy make the entire fabric of commerce location-independent in both space and time. From the perspective of any vendor in the electronic commerce space, it enables a transactional semantic for nearly any aspect of business involving authority or communication. For the consumer, it is the first way to be a player in the net using security skills commensurate with their prior experience yet not requiring their allegiance to any authority. For the merchant, it is yet another broadening of what constitutes commerce — and it is the successive calculus of these broadenings that explains why electronic commerce is indeed an idea whose time has come.

## 8. Conclusion

We believe that this notion of smartcard CAs is a new cryptographic primitive offering flexible application. Smartcard CAs are particularly valuable for electronic commerce, where they bring out the best of public key's benefits. We argue that overall, public key's advantages for commerce are less than they seem, because electronic commerce's purpose is cost-reduction, and public-key's security features do little to reduce per-transaction costs. However, smart-card CAs can help solve an open problem in electronic commerce, which has vexed designers of public key and symmetric key security alike. This problem is

large-scale, secure authorization. We suggest that most payments problems should really be viewed as authorization problems, and that this approach is profitable precisely because smartcard-issued certs lend themselves especially well to authorization.

## 9. Bibliography

[AnNe95]

R.J. Anderson and R.M. Needham, "Robustness Principles for Public-Key Protocols," *Advances in Cryptology - CRYPTO '95*, Springer-Verlag, Berlin, 1995.

[BlFeLa96]

M. Blaze, J. Feigenbaum and J. Lacy, "Decentralized Trust Management," Proc. IEEE Symp. on Security and Privacy, Oakland, May 1996.

[Ch92]

D. Chaum, "Achieving Electronic Privacy," *Scientific American*, August 1992, pp. 96-101.

[Da96]

D. Davis, "Compliance Defects in Public-Key Cryptography," Proc. 6th USENIX Security Symp., San Jose, July '96. pp. 171-178.

[DaSw92]

D. Davis and R. Swick, "Network Security via Private-Key Certificates," Proc. 3rd USENIX Security Symp., Baltimore, Sept. '92, pp. 239-242. Also in ACM Operating Systems Review, v.24, n.4, Oct. 1990.

[DeDoLeMa93]

T. Denny, B. Dodson, A.K. Lenstra and M.S. Manasse, "On the Factorization of RSA-120," in *Advances in Cryptology - CRYPTO '93*, Ed. by Douglas R. Stinson, 1994, Springer-Verlag Lecture Notes in Comp. Sci. #773, pp. 166-174.

[El96]

C. Ellison, "Establishing Identity Without Certification Authorities," Proc. 6th USENIX Security Symp., San Jose, July, 1996, pp. 67-76.

[GiPaStTr95]

D. Gifford, A. Payne, L. Stewart and W. Treese, "Payment Switches for Open Networks," Proc. 1st USENIX Workshop on Electronic Commerce, New York City, July 1995, pp. 69-75.

[LaAbBuWo91]

B. Lampson, M. Abadi, M. Burrows and E. Wobber, "Authentication in Distributed Systems: Theory and Practice," 13th ACM Symp. on Operating Systems Principles, Oct. 1991, pp. 165-182.

[NeMe95]

B.C. Neuman and G. Medvinsky, "Requirements for Network Payment: The NetCheque Perspective," Proc. IEEE Compcon '95, San Francisco, March 1995.

[Od96]

A. Odlyzko, "The Future of Integer Factoring," CryptoBytes, RSA Data Security, Inc., 1996.

[Vi90]

VISA International and Microsoft Corp., "Secure Transaction Technology Specifications," 1995.

## 10. Acknowledgements

The authors wish to thank the referees for their comments and Josh Lubarr, Henry Tumblin, Jack Rieden and other members of the Security Study Group at Open Market for their advice and support.

## 11. Availability

Patent pending; contact first author for queries of any sort.

# Smart Cards in Hostile Environments

Howard Gobioff  
*Carnegie Mellon Univ.*  
Pittsburgh, PA 15213  
hgoiboff@cs.cmu.edu

Sean Smith  
*IBM Research*  
Yorktown Heights, NY 10598  
sean@watson.ibm.com

J. D. Tygar  
*Carnegie Mellon Univ.*  
Pittsburgh, PA 15213  
tygar@cs.cmu.edu

Bennet Yee  
*UC San Diego*  
La Jolla, CA 92093  
bsy@cs.ucsd.edu

## Abstract

One often hears the claim that smart cards are the solution to a number of security problems, including those arising in point-of-sale systems. In this paper, we characterize the minimal properties necessary for the secure smart card point-of-sale transactions. Many proposed systems fail to provide these properties: problems arise from failures to provide secure communication channels between the user and the smart card while operating in a potentially hostile environment (such as a point-of-sale application.) Moreover, we discuss several types of modifications that can be made to give smart cards additional input/output capacity with a user, and describe how this additional I/O can address the hostile environment problem. We give a notation for describing the effectiveness of smart cards under various environmental assumptions. We discuss several security equivalences among different scenarios for smart cards in hostile environments. Using our notation, these equivalences include:

- private input  $\approx$  private output
- trusted input + one-bit trusted output  $\approx$  trusted output + one-bit trusted input
- secure input  $\approx$  secure output

## 1 Introduction

Point-of-sale (POS) systems introduce a number of security problems. In a traditional credit card

This research was supported in part by the Defense Advanced Research Projects Agency under contract F119628-93-C-0193, IBM, U.S. Department of Energy under Contract No. W-7405-ENG-36, the US Postal Service, and Visa International. Howard Gobioff was supported in part by a National Science Foundation Graduate Fellowship. Sean Smith performed this research at Los Alamos National Laboratory. The views and conclusions in this document are those of the authors and do not necessarily represent the official policies or endorsements of the US Government, its agencies, or any of the research sponsors.

model, the customer reveals his credit card number to the merchant. This allows a corrupt merchant to improperly use the customer's credit card.

To solve this problem, computer scientists have proposed the use of *smart cards* that can act as intermediate brokers. Smart cards are small handheld computational devices that can perform cryptographic operations. One type of smart card model is a *stored value card* containing an account balance register. The smart card is considered tamper-resistant, in that it is not feasible for any person to modify the smart card account balance without going through an approved protocol<sup>1</sup>. Many recent smart cards provide mechanisms that will cause any attempt to physically read data in the smart card to result in all data being zeroed (e.g., US Federal Information Processing Standard 140-1 [11].)

Similarly, the merchant's POS computer will contain a tamper-resistant register representing the merchant's account balance. When a customer makes a purchase, the smart card account balance is decremented by the amount of the purchase, and the merchant's POS account balance is incremented by the same amount. Later, smart cards and POS systems report their current account balances to a computer acting as a bank, and their accounts are accordingly modified. If the registers are truly tamper-proof, this approach appears to provide a safe way to exchange values off-line. This approach (with slight modifications) is taken in the proposed *MasterCard 2000* and *Visa Stored Value Card* systems [8, 10, 12].

A different set of approaches has been proposed by digital cash researchers [5, 6, 7]. "Electronic wallets" transfer an electronic token to the point-of-sale system. At a later time, the electronic token is "cashed in", for reconciliation, to the computer

<sup>1</sup>Tamper resistance is more difficult than it appears at first. Anderson and Kuhn[2] have show how to break a purportedly secure device. Kocher[9] has shown how to use timing attacks to discover RSA keys. And Boneh, DeMillo, and Lipton[4] have shown that a smart card performing the same encryption twice is vulnerable if an opponent can induce processor failures through a hostile environment (radiation, temperature extremes, etc.).

acting as a bank. Digital cash approaches typically provide anonymous transactions, and use fewer assumptions about tamper-resistance. In particular, Chaum [7] divides the smart card into an *observer*, a tamper-proof device trusted by the digital cash system, and the user's *representative*, in which the observer is embedded. The user has full control over the hardware embodying the representative, but has no internal access to the observer. The observer participates in Chaum's protocol and actively prevent double spending in such a way that the user need not trust the correctness of the observer with respect to leaking identity information; the observer may, however, cause denial of service.

However, both stored value cards and electronic wallets ignore one very feasible attack: since traditional smart cards do not contain any provision for directly displaying output or directly receiving input from the customer, they must depend on the merchant's POS system for I/O with the customer (this problem was observed in [3, 13, 14]). This introduces a significant vulnerability: for example, a corrupt merchant might try to charge the customer's smart card \$1000 for the purchase of a gold watch while truthfully reporting on his POS display that the purchase is for a \$10 watch battery. If the customer authorizes the smart card to transfer funds based on the displayed data, the merchant successfully defrauds the customer.

Note that the systemic threat that is being addressed by this paper differs dramatically from those being addressed by the above-mentioned observer model in digital cash systems. Here, we are concerned with the possibility of corruption of the POS terminal, so that the information displayed to the user — as part of an authorization request — shows one price, while the smart card is shown another. This variant of the Trojan Horse attack is impossible to solve without some way for the user to learn the true transaction value as seen by the smart card. In the observer model, Chaum assumes that the representative possesses secure I/O for communication with the user, a property not true of traditional smart cards.

This paper explores a number of variations in smart card designs that address this problem. We give an informal notation to describe equivalences of various smart card mechanisms to provide protection to interactions between the user and smart card where the smart card is accessed in a potentially hostile environment. These equivalences show that mechanisms that achieve certain security properties can be simulated by alternative mechanisms.

Further, we describe some potential designs for

smart cards with additional I/O channels direct to the user. For example, these designs<sup>2</sup> for smart cards contain LEDs that display values to be directly read by the smart card owner, or contain buttons to directly input material from the smart card owner. In this paper, we describe requirements for these I/O-enhanced smart cards and consequences of their theoretical security properties. However, we do not attempt here to discuss the physical construction, economics, or feasibility of various alternatives among these I/O-enhanced smart cards.

## 2 Our Model

We describe interactions between the smart card and the customer by separating the description of input and output. The security properties of both input and output can be described by the presence or absence of two attributes: privacy and trust.

*Privacy* means that the content of a communication cannot be observed by anyone who is neither the sender nor receiver. In our context, privacy refers to a customer – smart card communication being protected from observation by the merchant. If a communications channel is not private, we say it is *public*.

*Trust* means that a recipient has confidence in the origin and freshness of a communication. If the customer receives a trusted communication from the smart card, then he is confident that the communication originated from the smart card in the immediate past and is being received intact (for example, a multi-part message is being received unmodified by an adversary). If the customer has a trusted input channel to the smart card, the smart card can treat communications on that channel as fresh, in proper order, and having originated from the possessor of the smart card. If a communications channel is not trusted, we say it is *untrusted*.

If a communication is both trusted and private, we say it is *secure*.

### 2.1 Notation

We give informal definitions of our notation; this gives us a convenient shorthand for discussing security properties.

**A**  $\succ$  **B** This expression means that any protection mechanisms provided by a smart card with **B** can also be provided by a smart card with

---

<sup>2</sup>Please note there are some examples of smart cards that provide these I/O operations — such as the VISA/Toshiba Super Smart Card.

- A. For example, “trusted input  $\succ$  no input” because a smart card that has no input can be simulated by a smart card that does have trusted input.
- A  $\approx$  B** This expression means that both A  $\succ$  B and B  $\succ$  A.
- A + B** This expression refers to a class of communication security properties provided by smart cards that have both A and B. For example, “trusted input + trusted output  $\succ$  trusted input”, because any smart card that has trusted input only can be simulated by a smart card that has both trusted input and trusted output.

### 3 Methods

This section includes arguments showing the following equivalences:

- private input  $\approx$  private output
- trusted input + one-bit trusted output  $\approx$  trusted output + one-bit trusted input
- secure input  $\approx$  secure output

#### 3.1 Achieving these Properties

The most straightforward way for a customer to establish trusted and private communication channels with a smart card is to insert the smart card into a reader/writer device trusted by the customer. These devices directly provide *trusted* input and output, and with the proper physical shields (e.g., to prevent shoulder-surfing), also provide *private*, and hence *secure*, input and output.

In POS applications, the merchant controls the reader/writer. Indeed, from the customer’s point of view, the merchant’s smart card reader qualifies as a potentially hostile environment. Hence, we need to consider techniques to achieve trusted and private communication with smart cards in hostile environments.

One approach is to put a keypad and display directly on the smart card. Such peripherals increase the cost of the smart card (and may violate assumptions about the smart card’s physical security), but provide trusted communication for the customer; and, with physical shielding, can provide privacy as well. If keypads and displays on the smart card are infeasible, the customer could carry a trusted portable smart card reader [3]. In this sort of system, after the transaction parameters are transferred

to the smart card, the customer would transfer the card from the merchant’s terminal into the portable reader. Only if the customer approves the transaction would he move the card back to the merchant’s terminal. However, using portable readers this way may be unacceptably awkward for many POS applications; certainly, if the customers are willing to carry portable smart card I/O devices, we might as well omit the smart card and have the trusted I/O devices communicate directly with the merchant terminal via some higher bandwidth channel than a smart card (e.g., infra-red link or a cable).

Another approach routes communications through the merchant’s reader/writer, and protects those communications using information security techniques. For example, if the customer and the smart card shared knowledge of a large codebook, they could use this codebook to send messages to each other that intermediaries could neither understand nor forge. Alternatively, if the customer can perform cryptography in his head (such as digital signatures, or RSA or DES encryption) and can enter data using numeric keypads very quickly, then the customer and smart card could simply pass encrypted and/or signed messages to each other, achieving trust and, if encryption is used, privacy. However, doing operations with large codebooks from memory and performing RSA and DES encryptions in one’s head appears to be beyond the ability of most normal human beings.

#### 3.2 Equivalence

If a customer has direct secure input and output communication paths, he can safely communicate with his smart card and achieve safe POS applications. However, safe communications are possible over indirect, untrusted paths if the customer has a shared secret with the smart card, such as a one-time pad. Below we establish a set of equivalences and implications for various types of customer/smart card communication.

**Private input  $\succ$  private output.** If the customer has an input channel to the smart card that, should it actually reach the smart card, can be presumed private, the customer can turn an untrusted public output channel into an untrusted private output channel by giving the smart card a one-time pad to encrypt its output. (While methods using one-time pads may seem a bit far-fetched, in Section 4 below, we discuss a practical example.)

**Secure input  $\succ$  secure output.** Similarly, suppose the customer has a secure channel to the smart card. The customer can transform an untrusted public output channel into a secure channel by entering a one-time pad. Output from the smart card to the customer is encrypted with the one-time pad, including a data checksum to detect data integrity loss.

**Trusted input + one bit of trusted output  $\succ$  trusted output.** The customer can feed the value displayed by untrusted output back to the smart card. The smart card then uses its one bit of trusted output to signal that it received the value and agrees with it.

**Symmetry of input and output.** In a hostile environment, a symmetry exists between the customer and the smart card. (The principal differences are that the smart card has more memory and more computational ability.) Input from the customer's point of view is equivalent to output from the smart card's point of view, and *vice versa*. Suppose a rule exists of the form

$$\begin{aligned} X_1 \text{ input} + Y_1 \text{ output} &\succ \\ X_2 \text{ input} + Y_2 \text{ output} \end{aligned}$$

Then by this symmetry, we also have:

$$\begin{aligned} Y_1 \text{ input} + X_1 \text{ output} &\succ \\ Y_2 \text{ input} + X_2 \text{ output} \end{aligned}$$

Applying this rule to the above equivalences, we obtain the following:

**Private output  $\succ$  private input.** If the customer receives private output from the smart card, he can generate private input to the smart card. If the smart card presents a one-time pad to the customer through the private output, the customer can encrypt his desired input to the smart card (see Section 4 for an example).

**Secure output  $\succ$  secure input.** Similarly, we can use a one-time pad to guarantee the privacy of our communication. With a private channel, the smart card can present the customer with an authentication challenge. The customer provides an appropriate response and subsequent communication are encrypted by the one-time pad.

**Trusted output with one bit of trusted input  $\succ$  trusted input.** If the customer has trusted output from the smart card and one bit of trusted input, the customer can generate trusted input. (Note: if the customer can yank his smart card out of the merchant's reader/writer, then he has at least one bit of trusted input!) The customer provides his input to the smart card and the smart card echoes back the information to the customer. If the smart card echoes the wrong information, the customer uses the one bit of trusted input to inform the smart card of the communication failure. (This method uses an implicit assumption that the possessor of the smart card is an authorized user. By itself, this method does not provide protection against smart card theft.)

## 4 Achieving Secure Transactions

We discuss some possible requirements for an I/O-enhanced smart cards that would give a variety of security configurations. We make no attempt to discuss the technical or economic feasibility of I/O-enhanced smart cards; this paper is concerned with exploring various equivalences of security properties among different types I/O-enhanced smart cards.

Here are minimum requirements to accomplish a POS transaction: The customer must communicate to the smart card enough information to indicate the amount of the transaction. It is also necessary that the smart card know the merchant's identity so that it can verify it (in order to protect against Trojan horse attacks by untrusted POS terminals) — the merchant identity information is important to avoid problems later with unrolling transactions, e.g., in order to return defective or otherwise unsuitable merchandise. The customer need not personally provide the requisite information to the smart card. The merchant may provide the information directly to the smart card which will then verify it with the user through trusted input and output. The smart card does not require user authentication, which is why "trusted" means to possessor of card, and with trusted I/O privacy is not required for transaction authorization. If either trusted input or output is unavailable, then, as we see below, we may require additional privacy conditions.

From the customer's perspective, the only absolute requirement is to provide proper information to the smart card. The minimal required mechanism is trusted input. As we have seen, trusted input can be implemented through a variety of combinations

of input and output properties.

The merchant must be able to tell his POS system the amount of the expected transaction and know when the transaction completes. This requirement is satisfied if the merchant has trusted input to the POS system, which is trivial if the merchant controls the environment, and one bit of trusted output to indicate transaction completion.

## 4.1 Examples

**Private output  $\succ$  private input.** How can we get use private output to simulate private input? For example, if a smart card has an integral display unit but no direct input capabilities, customers can privately communicate with the smart card. One way is for the smart card to present a random sequence of digits on its display unit [1]. The customer then sends a sequence of increment, decrement, and next-digit commands to the smart card via the public, unsecure communication channel. These commands alter the smart card's initial random value until the smart card displays the input value desired by the customer. This is effectively a special form of a one-time pad. This approach can be used for both entering a password and transaction amounts. (Note that this method is vulnerable to an adversary that can simultaneously shoulder-surf the display and tap the input stream.)

**Private input  $\succ$  private output.** Conversely, how can we use private input to simulate private output? Consider a smart card containing an integral numeric keypad that but lacking human readable output capabilities. This smart card is inserted in a POS terminal providing (unsecure) output for the smart card. If the customer takes precautions to prevent observers from observing information typed into the smart card, the customer can provide the smart card with a one-time pad. The smart card could then encrypt data using the one-time pad. Since only the smart card and the customer are in possession of the one-time pad, they are the only parties able to read the message.

**Trusted input with one bit of trusted output  $\succ$  general trusted output.** A trusted input channel, such as a numeric keypad, allows the smart card to present trusted output over an untrusted communications channel. The customer feeds the output from the untrusted path back to the smart card via the trusted input channel. If the smart card receives an unexpected value from the customer, it

uses a single bit of trusted output, such as an integral LED, to signal the problem to the customer. (This method does not address the customer authentication problem for stolen smart cards.)

**Trusted output with one bit of trusted input  $\succ$  general trusted input.** If a smart card is capable of presenting trusted output to the customer (for example, through an integral display) and the customer can reply with one bit of trusted input (such as removing the smart card from the reader) then the customer and smart card can achieve a trusted input channel. The customer communicates to the smart card via an untrusted input channel and the smart card then echoes back the input message through the trusted output channel. If the customer disagrees with the message displayed by the smart card, the customer communicates this via the single bit of trusted input.

## 5 Conclusion

We believe that the corrupt point-of-sale terminal problem to be a major challenge for using smart cards in electronic commerce. We have begun a discussion of potential solutions by discussing equivalences among varying types of I/O-enhanced smart cards and the types of protection they provide.

We also believe that these mechanisms could also find applications outside of POS transactions. For example, consider the key management case: Imagine that a user has a portable device (such as a smart card) with a private key (for asymmetric) for electronically signing documents. How can the user make sure that his or her device only signs the document that he or she approved?

Our informal calculus of equivalences is meant to be suggestive instead of a formal reasoning method for smart card security. However, we believe that this notation could be formalized, and that the process of making it mathematically rigorous may illuminate further issues in the use of smart cards in hostile environments.

## References

- [1] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and delegation with smart-cards. Technical Report 67, DEC Systems Research Center, October 1990.
- [2] Ross Anderson and Markus Kuhn. Tamper resistance — a cautionary note. In *Proceedings of*

*The Second USENIX Workshop on Electronic Commerce, Oakland, CA, November 1996.*

- [3] Jean-Paul Boly, Antoon Bosselaers, Ronald Cramer, Rolf Michelsen, Stig Mjølsnes, Frank Muller, Torben Pedersen, Birgit Pfitzmann, Peter de Rooij, Berry Schoenmakers, Matthias Schunter, Luc Vallée, and Michael Waidner. The ESPRIT project CAFE — high security digital payment systems. In Dieter Gollmann, editor, *Computer Security — ESORICS 94, Third European Symposium on Research in Computer Security*, number 875 in Lecture Notes in Computer Science, Brighton, United Kingdom, November 1994. Springer-Verlag. ISBN 3-540-58618-0.
- [4] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. Cryptanalysis in the presence of hardware faults. *Personal Communication*, September 1996.
- [5] Stefan Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, Centrum voor Wiskunde en Informatica, 1993.
- [6] E. Brickell, P. Gemmell, and D. Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 457–466, 1995.
- [7] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.
- [8] Europay International, MasterCard, and Visa. Integrated circuit card specification for payment systems, October 1994.
- [9] P. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology: Crypto '96 Proceedings*, Lecture Notes in Computer Science. Springer-Verlag, 1996.
- [10] Mastercard launches development of smart card platform. Press Release, July 20, 1994.
- [11] U. S. National Institute of Standards and Technology. Federal information processing standards publication 140-1: Security requirements for cryptographic modules, January 1994.
- [12] Visa to develop and test prototype chip technology. Press Release, November 8, 1994.
- [13] Bennet Yee and Doug Tygar. Secure coprocessors in electronic commerce applications. In *Proceedings of The First USENIX Workshop on Electronic Commerce*, New York, New York, July 1995.
- [14] Bennet S. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.

# Analysis of the SSL 3.0 protocol

David Wagner

*University of California, Berkeley*  
daw@cs.berkeley.edu

Bruce Schneier

*Counterpane Systems*  
schneier@counterpane.com

## Abstract

The SSL protocol is intended to provide a practical, application-layer, widely applicable connection-oriented mechanism for Internet client/server communications security. This note gives a detailed technical analysis of the cryptographic strength of the SSL 3.0 protocol. A number of minor flaws in the protocol and several new active attacks on SSL are presented; however, these can be easily corrected without overhauling the basic structure of the protocol. We conclude that, while there are still a few technical wrinkles to iron out, on the whole SSL 3.0 is a valuable contribution towards practical communications security.

## 1 Introduction

The recent explosive growth of the Internet and the World Wide Web has brought with it a need to securely protect sensitive communications sent over this open network. The SSL 2.0 protocol has become a de facto standard for cryptographic protection of Web `http` traffic. But SSL 2.0 has several limitations—both in cryptographic security and in functionality—so the protocol has been upgraded, with significant enhancements, to SSL 3.0. This new version of SSL will soon see widespread deployment. The IETF Transport Layer Security working group is also using SSL 3.0 as a base for their standards efforts. In short, SSL 3.0 aims to provide Internet client/server applications with a practical, widely-applicable connection-oriented communications security mechanism.

This note analyzes the SSL 3.0 specification [FKK96], with a strong focus on its cryptographic security. We assume familiarity with the SSL 3.0 specification. Explanations of some of the cryptographic concepts can be found in [Sch96].

The paper is organized as follows. Section 2 briefly

gives some background on SSL 3.0 and its predecessor SSL 2.0. Sections 3 and 4 explore several possible attacks on the SSL protocol and offer some technical discussion on the cryptographic protection afforded by SSL 3.0; this material is divided into two parts, with the SSL record layer analyzed in Section 3 and the SSL key-exchange protocol considered in Section 4. Finally, Section 5 concludes with a high-level view of the SSL protocol’s strengths and weaknesses.

## 2 Background

SSL is divided into two layers, with each layer using services provided by a lower layer and providing functionality to higher layers. The SSL record layer provides confidentiality, authenticity, and replay protection over a connection-oriented reliable transport protocol such as TCP. Layered above the record layer is the SSL handshake protocol, a key-exchange protocol which initializes and synchronizes cryptographic state at the two endpoints. After the key-exchange protocol completes, sensitive application data can be sent via the SSL record layer.

SSL 2.0 had many security weaknesses which SSL 3.0 aims to fix. We briefly describe a short list of the flaws in SSL 2.0 which we have noticed. In export-weakened modes, SSL 2.0 unnecessarily weakens the authentication keys to 40 bits. SSL 2.0 uses a weak MAC construction, although post-encryption seems to stop attacks. SSL 2.0 feeds padding bytes into the MAC in block cipher modes, but leaves the padding-length field unauthenticated, which may potentially allow active attackers to delete bytes from the end of messages. There is a ciphersuite rollback attack, where an active attacker edits the list of ciphersuite preferences in the hello messages to invisibly force both endpoints to use a weaker form of encryption than they otherwise would choose; this serious flaw limits SSL 2.0’s strength to “least common denominator” security when active attacks are a threat. Oth-

ers have also discovered some of these weaknesses: Dan Simon independently pointed out the ciphersuite rollback attack, Paul Kocher has addressed these concerns [Koc96], and the PCT 1.0 protocol [PCT95] discussed and countered some (though not all) of these flaws.

## 3 The record layer

This section considers the cryptographic strength of the record layer protocol, and assumes that the key-exchange protocol has securely set up session state, keys, and security parameters. Of course, a secure key-exchange protocol is vital to the security of application data, but an examination of attacks on the SSL key-exchange protocol is postponed until the next section.

The SSL record layer addresses fairly standard problems that have received much attention in the cryptographic and security literature [KV83], so it is reasonable to hope that SSL 3.0 provides fairly solid protection in this respect. As we shall see, this is not far from the truth. We consider confidentiality and integrity protection in turn.

### 3.1 Confidentiality: eavesdropping

The SSL protocol encrypts all application-layer data with a cipher and short-term session key negotiated by the handshake protocol. A wide variety of strong algorithms used in standard modes is available to suit local preferences; reasonable applications should be able to find an encryption algorithm meeting the required level of security, US export laws permitting. Key-management is handled well: short-term session keys are generated by hashing random per-connection salts and a strong shared secret. Independent keys are used for each direction of a connection as well as for each different instance of a connection. SSL will provide a lot of known plaintext to the eavesdropper, but there seems to be no better alternative; since the encryption algorithm is required to be strong against known-plaintext attacks anyway, this should not be problematic.

### 3.2 Confidentiality: traffic analysis

When the standard attacks fail, a cryptanalyst will turn to more obscure ones. Though often maligned, traffic analysis is another passive attack worth con-

sidering. Traffic analysis aims to recover confidential information about protection sessions by examining unencrypted packet fields and unprotected packet attributes. For example, by examining the unencrypted IP source and destination addresses (and even TCP ports), or examining the volume of network traffic flow, a traffic analyst can determine what parties are interacting, what type of services are in use, and even sometimes recover information about business or personal relationships. In practice, users typically consider the threat of this kind of coarse-grained tracking to be relatively harmless, so SSL does not attempt to stop this kind of traffic analysis. Ignoring coarse-grained traffic analysis seems like a reasonable design decision.

However, there are some more subtle threats posed by traffic analysis in the SSL architecture. Bennet Yee has noted that examination of ciphertext lengths can reveal information about URL requests in SSL- or SSL-encrypted Web traffic [Yee96]. When a Web browser connects to a Web server via an encrypted transport such as SSL, the GET request containing the URL is transmitted in encrypted form. Exactly which Web page was downloaded by the browser is clearly considered confidential information—and for good reason, as knowledge of the URL is often enough for an adversary to obtain the entire Web page downloaded—yet traffic analysis can recover the identity of the Web server, the length of the URL requested, and the length of the html data returned by the Web server. This leak could often allow an eavesdropper to discover what Web page was accessed. (Note that Web search engine technology is certainly advanced enough to catalogue the data openly available on a Web server and find all URLs of a given length on a given server which return a given amount of html data.)

This vulnerability is present because the ciphertext length reveals the plaintext length.<sup>1</sup> SSL includes support for random padding for the block cipher modes, but not for the stream cipher modes. We believe that SSL should at the minimum support the usage of random-length padding for all cipher modes, and should also strongly consider requiring it for certain applications.

<sup>1</sup>This is strictly speaking only true of stream ciphers, but they are currently the common case. With block ciphers, plaintexts are padded out to the next 8-byte boundary, so one can only recover a close estimate of the plaintext length.

### 3.3 Confidentiality: active attacks

It is important that SSL securely protect confidential data even against active attacks. Of course, the underlying encryption algorithm should be secure against adaptive chosen-plaintext/chosen-ciphertext attacks, but this is not enough on its own. Recent research motivated by the IETF ipsec (IP security) working group has revealed that sophisticated active attacks on a record layer can breach a system's confidentiality even when the underlying cipher is strong [Bel96]. It appears that the SSL 3.0 record layer resists these powerful attacks; it is worth discussing in some depth why they are foiled.

One important active attack on ipsec is Bellovin's cut-and-paste attack [Bel96]. Recall that, to achieve confidentiality, link encryption is not enough—the receiving endpoint must also guard the sensitive data from inadvertent disclosure. The cut-and-paste attack exploits the principle that most endpoint applications will treat inbound encrypted data differently depending on the context, protecting it more assiduously when it appears in some forms than in others.<sup>2</sup> The cut-and-paste attack also takes advantage of a basic property of the cipher-block chaining mode: it recovers from errors within one block, so transplanting a few consecutive ciphertext blocks between locations within a ciphertext stream results in a corresponding transfer of plaintext blocks, except for a one-block error at the beginning of the splice. In more detail, Bellovin's cut-and-paste attack cuts an encrypted ciphertext from some packet containing sensitive data, and splices it into the ciphertext of another packet which is carefully chosen so that the receiving endpoint will be likely to inadvertently leak its plaintext after decryption. For example, if cut-and-paste attacks on the SSL record layer were feasible, they could be used to compromise site security: a cut-and-paste attack on a SSL server-to-client Web page transfer could splice ciphertext from a sensitive part of that html transfer into the hostname portion of a URL included elsewhere in the transferred Web page, so that when a user clicks on the booby-trapped URL link his browser would interpret the decryption of the spliced sensitive ciphertext as a hostname and send a DNS domain name lookup for

it in the clear, ready for capture by the eavesdropping attacker. Cut-and-paste attacks, in short, enlist the unsuspecting receiver to decrypt and inadvertently leak sensitive data for them.

SSL 3.0 stops cut-and-paste attacks. One partial defense against cut-and-paste attacks is to use independent session keys for each different context. This prevents cutting and pasting between different connections, different directions of a connection, etc. SSL already uses independent keys for each direction of each incarnation of each connection. Still, cutting and pasting within one direction of a transfer is not prevented by this mechanism. The most comprehensive defense against cut-and-paste attacks is to use strong authentication on all encrypted packets to prevent enemy modification of the ciphertext data. The SSL record layer does employ this defense, so cut-and-paste attacks are completely foiled. For a more complete exposition on cut-and-paste attacks, see Bellovin's paper [Bel96].

The short-block attack is another active attack against ipsec which can be found in Bellovin's paper [Bel96]. The short-block attack was originally applied against DES-CBC ipsec-protected TCP data when the final message block contains a short one-byte plaintext and the remainder of it is filled by random padding. One guesses at the unknown plaintext byte by replacing the final ciphertext block with another ciphertext block from a known plaintext/ciphertext pair. Correct guesses can be recognized by the validity of the TCP checksum: an incorrect guess will cause the packet to be silently dropped by the receiver's TCP stack, but the correct guess will cause a recognizable ACK to be returned. Knowledge of the corresponding plaintext for a correctly guessed replacement ciphertext block enables the enemy to recover the unknown plaintext byte. Because the receiving ipsec stack ignores the padding bytes, the short-block attack requires about  $2^8$  known plaintexts and  $2^8$  active online trials to recover such an unknown trailing byte. Many distracting technicalities have been significantly simplified; see Bellovin's paper [Bel96] for more details.

There are no obvious short-block attacks on SSL. The SSL record layer format is rather similar to the old vulnerable ipsec layout, so it is admittedly conceivable that a modified version of the attack might work against SSL. In any case, standard SSL-encrypting Web servers probably would not be threatened by a short-block type of attack, since they do not typically encrypt short blocks. (Note, however, that a SSL-encrypting telnet client should

<sup>2</sup>In the ipsec world, encrypted data to TCP user ports is not protected by the operating system nearly as strongly as encrypted data to the system TCP login or telnet port. For a SSL-protected Web connection, the client browser will guard the path portion of a URL more carefully than the hostname portion, as the hostname portion may subsequently appear unencrypted in DNS queries and IP source addresses, whereas the path portion of a URL is encrypted via SSL.

demand particularly robust protection against short-block attacks, as each keystroke is typically sent in its own one-byte-long packet.)

In summary, our analysis did not uncover any active attacks on the confidentiality protection of the SSL 3.0 record layer.

### 3.4 Message authentication

In addition to protecting the confidentiality of application data, SSL cryptographically authenticates sensitive communications. On the Internet, active attacks are getting easier to launch every day. We are aware of at least two commercially available software packages to implement active attacks such as IP spoofing and TCP session hijacking, and they even sport a user-friendly graphical interface. Moreover, the financial incentive for exploiting communications security vulnerabilities is growing rapidly. This calls for strong message authentication.

SSL protects the integrity of application data by using a cryptographic MAC. The SSL designers have chosen to use HMAC, a simple, fast hash-based construction with some strong theoretical evidence for its security [BCK96]. In an area where several initial ad-hoc proposals for MACs have been cryptanalyzed, these provable security results are very attractive. HMAC is rapidly becoming the gold standard of message authentication, and it is an excellent choice for SSL. Barring major unexpected cryptanalytic advances, it seems unlikely that HMAC will be broken in the near future.

We point out that SSL 3.0 uses an older obsolete version of the HMAC construction. SSL should move to the updated current HMAC format when convenient, for maximal security.

On the whole, SSL 3.0 looks very secure against straightforward exhaustive or cryptanalytic attacks on the MAC. SSL 2.0 had a serious design flaw in that it used an insecure MAC—though post-encryption saved this from being a direct vulnerability—but SSL 3.0 has fixed this mistake. The SSL MAC keys contain at least 128 bits of entropy, even in export-weakened modes, which should provide excellent security for both export-weakened and domestic-grade implementations. Independent keys are used for each direction of each connection and for each new incarnation of a connection. The choice of HMAC should stop cryptanalytic attacks. SSL does not provide non-repudiation services, and it seems reasonable to deliberately leave that to spe-

cial higher-level application-layer protocols.

### 3.5 Replay attacks

The naive use of a MAC does not necessarily stop an adversary from replaying stale packets. Replay attacks are a legitimate concern, and as they are so easy to protect against, it would be irresponsible to fail to address these threats. SSL protects against replay attacks by including an implicit sequence number in the MACed data. This mechanism also protects against delayed, re-ordered, or deleted data. Sequence numbers are 64 bits long, so wrapping should not be a problem. Sequence numbers are maintained separately for each direction of each connection, and are refreshed upon each new key-exchange, so there are no obvious vulnerabilities.

### 3.6 The Horton principle

Let's recall the ultimate goal of message authentication. SSL provides message integrity protection just when the data passed up from the receiver's SSL record layer to the protected application exactly matches the data uttered by the sender's protected application to the sender's SSL record layer. This means, approximately, that it is not enough to apply a secure MAC to just application data as it is transmitted over the wire—one must also authenticate any context that the SSL mechanism depends upon to interpret inbound network data. For lack of a better word, let's call this “the Horton principle” (with apologies to Dr. Seuss) of semantic authentication: roughly speaking we want SSL to

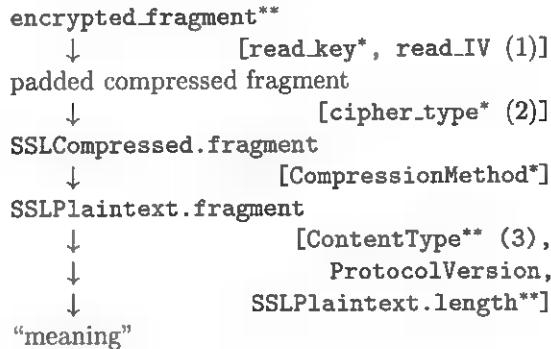
“authenticate what was meant, not what was said.”

To phrase it another way,

Eschew unauthenticated security-critical context.

SSL 2.0 suffered from at least one flaw along these lines: it included padding data but not the length of the padding in the MAC input, so an active attacker could manipulate the cleartext padding length field to compromise message integrity. An analysis checking SSL 2.0's compliance with the Horton principle would have uncovered this flaw; therefore, we undertake an informal analysis of SSL 3.0 following the guidelines of the Horton principle.

Figure 1: Analysis of security-critical context



Notes:

- \* session state synchronized by the key-exchange protocol.
- \*\* protected by the MAC.

- (1) `read_IV` is initially taken from the session state, then taken from the last ciphertext block of the previous `encrypted_fragment`.
- (2) for block ciphers, padding is removed from the end of the padded fragment.

The SSL record layer depends on a lot of context to interpret, decrypt, decompress, de-multiplex, and dispatch data from the wire. It is instructive to follow the chain of this processing of inbound network data, catalogue all the security-critical context which this processing depends on, and check to ensure that the critical context has been authenticated. This ensures that we have applied the MAC properly to all security-relevant items and fulfilled the Horton principle. Because the `encrypted_fragment` field is authenticated by the MAC, we will assume that that field is trustworthy, and follow its transformation into application data (“meaning”). The right-justified bracketed items in Figure 1 identify security-critical context used in each step of processing.

Figure 1 indicates that SSL 3.0 follows the Horton principle fairly closely. One minor exception is that the integrity of the `ProtocolVersion` field is not protected. (We refer specifically to the `SSLCiphertext.ProtocolVersion` field in the record layer, not the `ClientHello.client_version` field from the handshake protocol; the latter is protected, but the former is not.) If the `ProtocolVersion` field is ever used by SSL, it should

be authenticated; if not, it should not be present in the packet format. Also, it is worth mentioning that the final result of the inbound processing is a stream of bytes from the application data stream, and message boundaries are not preserved. Any application that relies on message boundaries—such as a UDP-based program—will have to impose a higher-layer message length protocol on top of SSL. On the whole, though, our “Horton principle”-inspired analysis revealed no major weaknesses, to SSL 3.0’s credit.

### 3.7 Summary

In summary, the protection of application data by the SSL record layer is, on the whole, quite good. The preceding section indicated a few small areas of concern, but they should be considered minor and the exception to the rule.

## 4 The key-exchange protocol

This section considers the security of the SSL handshake protocol as well as other SSL meta-data transport. The design of a secure key-exchange protocol is a thorny endeavor. There is a significant amount of complexity involved, so the discovery of a few weaknesses should not prove surprising. The following analysis describes a number of shortcomings of the SSL meta-data protection mechanisms, mostly in areas that have seen recent changes. The SSL 3.0 key-exchange protocol appears to be a significant advance over SSL 2.0, but it still bears a few scars from growing pains.

### 4.1 Overview of the handshake flow

The SSL 3.0 handshake-protocol message flow involves client and server negotiating a common ciphersuite acceptable to both parties, exchanging random nonces, and the client sending an encrypted `master_secret`. Then each verifies that their protocol runs match by authenticating all messages with the `master_secret`, and assuming that the check succeeds, both generate session keys from the `master_secret` and proceed to send application data. The SSL protocol also includes a more lightweight session resumption protocol which allows two parties who have already exchanged a `master_secret` to generate updated session keys and start a new connection with those parameters.

## 4.2 Ciphersuite rollback attacks

The SSL 2.0 key-exchange protocol contained a serious flaw: an active attacker could silently force a domestic user to use export-weakened encryption, even if both endpoints supported and preferred stronger-grade algorithms. This is known as a ciphersuite rollback attack, and it can be performed by editing the cleartext list of supported ciphersuites sent in **hello** messages. SSL 3.0 fixes this vulnerability by authenticating all the handshake protocol messages with the **master secret**, so such enemy tampering can be determined at the end of the handshake and the session terminated if necessary.

We describe the SSL 3.0 mechanism for preventing modification of handshake protocol messages in more detail. There are several generic vulnerabilities in this part of the SSL handshake protocol, so some introduction is in order. All the initial handshake protocol messages are sent, unprotected, in the clear. Instead of modifying the parameters in use at the moment, the key-exchange protocol modifies a pending session state. After the negotiation is complete, each party sends a short **change cipher spec** message, which simply alerts the other to upgrade the status of the pending session state to current. The new session state is used starting with the next message, though the **change cipher spec** message is unprotected.<sup>3</sup> Immediately following the **change cipher spec** comes the **finished** message, which contains a MAC on all the handshake protocol messages keyed by the **master secret**. (For peculiar non-security reasons, the **change cipher spec** and alert messages are not authenticated in the **finished** message.) The 48-byte **master secret** is never disclosed; instead, session keys are generated from it. This ensures that even if the session keys are recovered, the **master secret** will remain secret, so the handshake protocol messages will be securely authenticated. The **finished** message is itself protected with the newly established ciphersuite. Neither party is supposed to accept application data until it has received and verified a **finished** message from the other party.

## 4.3 Dropping the change cipher spec message

One quirk of the SSL key-exchange protocol is that the **change cipher spec** message is not protected by the message authentication in the **finished** message. This can potentially allow the cryptanalyst to get a foot in the door. We recall the normal SSL message flow:

```
...  
1.  $C \rightarrow S$  : [change cipher spec]  
2.  $C \rightarrow S$  : [finished:]  $\{a\}_k$   
3.  $S \rightarrow C$  : [change cipher spec]  
4.  $S \rightarrow C$  : [finished:]  $\{a\}_k$   
5.  $C \rightarrow S$  :  $\{m\}_k$   
...
```

where  $\{\cdot\}_k$  represents the keyed cryptographic transforms used by the record layer,  $m$  denotes a plaintext message sent after the key-exchange is finished, and  $a$  represents the **finished** message's authentication code, which is obtained by computing a symmetric MAC on the previous handshake messages (excluding the **change cipher spec** message). Note that before the receipt of a **change cipher spec** message, the current ciphersuite offers no encryption or authentication and the pending ciphersuite includes the negotiated ciphersuite; upon receiving a **change cipher spec** message, implementations are supposed to copy the pending ciphersuite to the current ciphersuite and enable cryptographic protection in the record layer.

We describe an attack that takes advantage of the lack of protection for **change cipher spec** messages. We assume the special case where the negotiated ciphersuite includes only message authentication protection and no encryption. The active attacker intercepts and deletes the **change cipher spec** messages, so that the two endpoints never update their current ciphersuite; in particular, the two endpoints never enable message authentication or encryption in the record layer for incoming packets. Now the attacker allows the rest of the interaction to proceed, stripping off the record layer authentication fields from **finished** messages and session data. At this point there is no authentication protection for session data in effect, and the active attacker can modify the transmitted session data at will. The impact is that, when an authentication-only transform is negotiated, an active attacker can defeat the authentication protection on session data, transparently causing both parties to accept incoming session data without any cryptographic integrity protection.

<sup>3</sup>More precisely, it is protected with the old session state, which initially is set up to provide no protection. The discussion ignores the complicating case of a handshake protocol execution which changes cryptographic parameters on a connection that already has some protection in effect.

We summarize the attack flow:

```

...
1. C → M : [change cipher spec]
2. C → M : [finished:] {a}k
2'. M → S : [finished:] a
3. S → M : [change cipher spec]
4. S → M : [finished:] {a}k
4'. M → C : [finished:] a
5. C → M : {m}k
5'. M → S : m
...

```

Remember, in this flow  $\{m\}_k$  denotes the transmission of a message  $m$  along with a message authentication field keyed by  $k$ ; given  $\{m\}_k$  it is easy to strip off the MAC field and recover  $\{m\}$ , since no encryption is in use here. Note that the attacker can easily replace the unprotected session data  $m$  in flow 5' by forged data of his choice.

It is worth pointing out what happens when the negotiated ciphersuite includes encryption. Then the client's **finished** message is sent encrypted, but the server expects to receive it unencrypted, so it does not suffice to strip off the MAC field—instead, the attacker must recover the encryption key  $k$  and decrypt  $\{a\}_k$  to obtain  $a$ . Therefore the attack will be foiled when the negotiated ciphersuite includes strong encryption. In the intermediate case where weak encryption (such as a 40-bit exportable mode) is used, the attacker may be able to carry out this attack if it is possible to perform an online exhaustive keysearch to recover the short encryption key.<sup>4</sup> In all fairness, real-time online exhaustive keysearch of a 40-bit cipher is currently out of reach for many adversaries, although advances in computation power may make it a more serious threat in the future.

The simplest fix is to require that a SSL implementation receive a **change cipher spec** message before accepting a **finished** message. Some readers might complain that this requirement ought to be obvious with a moment's reflection, even if it is not explicitly

---

<sup>4</sup>A note about the amount of known plaintext available is in order. When a block cipher mode (such as 40-bit RC2 or 40-bit DES) is in use, there will be 4 bytes of known plaintext in the header of the **finished** message and another 4–8 bytes in the padding fields, so enough known text is available. For unpadded 40-bit stream cipher modes, there is only the 4 bytes of known plaintext in the **finished** message header; if the client immediately sends encrypted session data after sending the **finished** message (as is allowed in Section 7.6.9 of the SSL 3.0 specification) then enough additional known plaintext will probably be available to uniquely recover the stream cipher key; otherwise, about  $2^8$  possible 40-bit keys will be suggested, and the attacker must settle for a  $2^{-8}$  chance of success.

stated in the SSL specification. We cannot fault such clarity of vision. However, we settle for the observation that at least one implementation has fallen for this pitfall. After performing the theoretical analysis, we examined Netscape's SSLRef 3.0b1 reference source code for SSL 3.0. Indeed, the necessary check is not made there; though we have not actually implemented the attack, it appears that SSLRef 3.0b1 will fall to a **change cipher spec** dropping attack when an authentication-only ciphersuite is negotiated.

A more radical fix would include the **change cipher spec** message in the the **finished** message's message authentication calculation. This would require a change to the SSL specification; however, it also would have the advantage of being more robust in face of implementation flaws.

At the least, we recommend that future SSL documents include a warning about this pitfall. Explicitness is a virtue.

#### 4.4 Key-exchange algorithm rollback

The SSL 3.0 handshake protocol also contains another design flaw. A server can send short-lived public key parameters, signed under its long-term certified signing key, in the **server key exchange** message. Several key-exchange algorithms are supported, including ephemeral RSA and Diffie-Hellman public keys. Unfortunately, the signature on the short-lived parameters does not protect the field which specifies which type of key-exchange algorithm is in use. Note that this violates the Horton principle: SSL should sign not just the public parameters but also all data needed to interpret those parameters.

For convenience, we reprint the relevant SSL 3.0 data structures from the the **server key exchange** message here.

```

enum { rsa, diffie_hellman, ... }
    KeyExchangeAlgorithm;
struct {
    opaque rsa_modulus;
    opaque rsa_exponent;
} ServerRSAParams;
struct {
    opaque dh_p;
    opaque dh_g;
    opaque dh_Ys;
} ServerDHParams;
struct {
    select (KeyExchangeAlgorithm) {

```

```

    case diffie_hellman:
        ServerDHParams params;
        Signature signed_params;
    case rsa:
        ServerRSAParams params;
        Signature signed_params;
    }
} ServerKeyExchange;

```

The `signed_params` field contains the server's signature on a hash of the relevant `ServerParams` field, but the signature does *not* cover the `KeyExchangeAlgorithm` value. Therefore, by modifying the (unprotected) `KeyExchangeAlgorithm` field, we can abuse the server's legitimate signature on a set of Diffie-Hellman parameters and fool the client into thinking the server signed a set of ephemeral RSA parameters.

We should point out that particularly cautious implementation might not be fooled by such tricks, if they check the length of the `ServerParams` field carefully. For example, SSLRef 3.0b1 is paranoid enough that it would detect such an attack. However, in general, the specification is silent on the matter, and some compliant implementations could easily be vulnerable.

If the implementation can be fooled, an active attack can be constructed. Perform a ciphersuite rollback attack to coerce the server into using the ephemeral Diffie-Hellman key exchange algorithm. Modify the `server key exchange` message, changing the `KeyExchangeAlgorithm` field to select ephemeral RSA key-exchange but leaving the `ServerParams` field and the server's signature untouched. Unless implementors are exceptionally foresighted or paranoid, the server's Diffie-Hellman prime modulus  $p$  (`dh_p`) and generator  $g$  (`dh_g`) will probably be interpreted by the client as a correctly signed short-lived RSA modulus  $p$  (`rsa_modulus`) with exponent  $g$  (`rsa_exponent`). Watch as the client encrypts the `pre_master_secret` with the bogus RSA values. Intercept the RSA encrypted value  $k^g \bmod p$ ; recover  $k$ , the PKCS encoding of the `pre_master_secret`, by taking  $g$ -th roots, which can be done efficiently since  $p$  is prime. Now that the `pre_master_secret` is compromised, it is easy to spoof the rest of the key exchange, including forging `finished` messages, to both endpoints. Thereafter one can decrypt all the sensitive application data transmitted or forge fake data on that SSL connection; all cryptographic protection has been wholly defeated.

We summarize this attack in the following attack flow

(omitting many irrelevant fields and messages):

```

[client hello:]
1. C → M : SSL_RSA....
1'. M → S : SSL_DHE_RSA....
[server hello:]
2. S → M : SSL_DHE_RSA....
2'. M → C : SSL_RSA....
[server key exchange:]
3. S → M : {p,g,y}_{K_S},diffie_hellman
3'. M → C : {p,g,y}_{K_S},rsa
[client key exchange:]
4. C → M : k^g mod p
4'. M → S : g^x mod p
...

```

At the end of the key-exchange, the client's value of the `pre_master_secret` is  $k$ , while the server's value is  $g^{xy} \bmod p$  where  $x$  was chosen by the attacker  $M$ ; of course, both of these are known to the attacker  $M$ , and all secrets are derived from these values, so all subsequent cryptographic transforms offer no protection against  $M$ .

## 4.5 Replay attacks on anonymous key-exchange

There is another similar, but lesser, design flaw in the `server key exchange` message. The standard key-exchange algorithms bind the signature on the short-lived cryptographic parameters to the connection by hashing with server and client nonces, but due to some oversight, anonymous key-exchanges do not perform this binding. Therefore, if one can convince a server to perform one anonymous key-exchange, then one will be able to spoof the server in all future sessions that use anonymous key-exchanges. Any client that will accept an anonymous key-exchange (even if it does not suggest it in the `client hello` ciphersuite negotiation) is then vulnerable to such spoofing. To stop this attack, the server's signature on the anonymous key-exchange parameter should indicate that the server is willing to accept anonymous key exchange and be vulnerable to man-in-the-middle attacks, and this signature should be bound to the current session. This requires binding that signature to the connection-specific random nonces.

For clarity, we reprint the relevant SSL data structures:

```

digitally-signed struct {
    select (SignatureAlgorithm) {

```

```

    case anonymous: struct { };
    case rsa:
        opaque md5_hash[16];
        ...
    }
} Signature;
md5_hash = MD5(ClientHello.random
+ ServerHello.random + ServerParams);

```

Note that, in the case of an anonymous key-exchange, the signature is over an empty structure; the signature does not include `ClientHello.random` or `ServerHello.random` and thus is not bound to the current session. Therefore, once an attacker has collected one anonymous `Signature` structure from a server, the attacker can spoof the server in future sessions and replay the old `Signature` structure without detection.

The key-exchange algorithm rollback attack and this replay attack serve to illustrate the dangers of a flexible ciphersuite negotiation algorithm. It is possible to end up with “least common denominator security”, where SSL is only as secure as the weakest key exchange algorithm (or weakest ciphersuite) supported.

## 4.6 Version rollback attacks

SSL 3.0 implementations will likely be flexible enough to accept SSL 2.0 connections, at least in the short-term. This threatens to create the potential for version rollback attacks, where an opponent modifies the `client hello` to look like a SSL 2.0 hello message and proceeds to exploit any of the numerous SSL 2.0 vulnerabilities.

Paul Kocher designed a fascinating strategy to detect version rollback attacks on SSL 3.0. Client implementations which support SSL 3.0 embed some fixed redundancy in the (normally random) RSA PKCS padding bytes to indicate that they support SSL 3.0. Servers which support SSL 3.0 will refuse to accept RSA-encrypted key-exchanges over SSL 2.0-compatibility connections if the RSA encryption includes those distinctive non-random padding bytes. This ensures that a client and server which both support SSL 3.0 will be able to detect version rollback attacks which try to coerce them into using SSL 2.0. Moreover, old SSL 2.0 clients will be using random PKCS padding, so they will still work with servers that support SSL 2.0.

The goal is to detect when both sides support SSL 3.0 even in the face of active attacks. Paul Kocher’s

countermeasure is a very clever approach. However, there are still a few little vulnerabilities to be worked out.

First, the success of the countermeasure depends vitally on the assumption that SSL 2.0 will only support RSA key-exchange; non-RSA public key-exchange algorithms will not admit the special padding redundancy, so version rollback attacks can not be detected if the server supports non-RSA key-exchange methods while operating in SSL 2.0 mode. In fact, this assumption can be violated in servers which support both SSL 2.0 and SSL 3.0. The SSL 3.0 specification optionally allows servers which support SSL 2.0 backwards compatibility to accept SSL 3.0 ciphersuites in SSL 2.0 `client hello` messages.

We illustrate a possible attack flow on such a server:

```

[client hello:]
1. C → M : v3.0, SSL_RSA....
1'. M → S : v2.0, SSL_DHE....
[server hello:]
2. S → C : v2.0, SSL_DHE....
[server key exchange:]
3. S → C : {p, g, y}KS, diffie_hellman
[client key exchange:]
4. C → S : {p, g, x}, diffie_hellman
...

```

Here two endpoints which support both SSL 3.0 and SSL 2.0 have been transparently forced to revert to the SSL 2.0 protocol. Moreover, neither endpoint is allowed to learn that the other endpoint supports SSL 3.0, since RSA key-exchange has been avoided. Now the active attacker can exploit any of the many attacks on SSL 2.0, such as ciphersuite rollback attacks, taking advantage of the weak message authentication found in SSL 2.0, etc. This is not good.

Second, a potential vulnerability arises from mixing SSL versions across resumed sessions. The specification does not forbid or discourage SSL 2.0-compatible SSL servers from accepting a SSL 2.0 `client hello` request to resume a session which was originally initiated with SSL 3.0. After resuming a SSL 3.0-created session with SSL 2.0, the attacker is free to perform any of the numerous SSL 2.0 active attacks, such as ciphersuite rollback. If the original SSL 3.0 session included client authentication, this allows an attacker to spoof the client: the attacker rolls back to SSL 2.0 via session resumption, rolls back to an export-weakened ciphersuite, exhaustively recovers the 40 bit key through brute force, and takes advantage of the SSL 2.0 weakness that MAC keys are 40 bits long in export-weakened modes to forge data and spoof the victim client to the server.

We illustrate the first attack flow:

```
[client hello:]  
1. C → S : v3.0, create new session  
[server hello:]  
2. S → C : v3.0, created  
  
[client hello:]  
3. M → S : v2.0, resume previous session  
[server hello:]  
4. S → M : v2.0, resumed
```

In the more sophisticated second attack, the attacker replaces messages 3 and 4 with a ciphersuite rollback attack on SSL 2.0:

```
[client hello:]  
3'. M → S : v2.0, resume previous session,  
      SSL_RC4_128_EXPORT40_WITH_MD5  
[server hello:]  
4'. S → M : v2.0, resumed,  
      SSL_RC4_128_EXPORT40_WITH_MD5  
[M recovers 40-bit MAC and RC4 keys k]  
5'. M → S : {m}_k  
6'. S → M : {m'}_k
```

Here  $m$  represents fake session data chosen by  $M$ ; recall that SSL 2.0 used 40-bit MAC keys in export-weakened modes, so a simple 40-bit exhaustive search recovers all the record layer keys and lets  $M$  forge the authentication and encryption. The server accepts the forgery  $\{m\}_k$  as a valid message. The server may also send back a confidential response  $\{m'\}_k$ , and of course  $M$  can decrypt and recover the plaintext  $m'$ . Therefore all cryptographic protection has been compromised in this scenario.

These attacks do not necessarily represent a fundamental limitation of SSL backwards compatibility. It seems that they can be fixed with minor changes to the specification: servers supporting both SSL 2.0 and SSL 3.0 should not let clients mix SSL versions across session resumption and should not support non-RSA key-exchange algorithms in SSL 2.0 compatibility mode. In any case, the right long-term fix is for servers to stop accepting SSL 2.0 connections.

#### 4.7 Safeguarding the master\_secret

Ensuring that the `master_secret` remains truly secret is tremendously important to the security of SSL. All session keys are generated from the `master_secret`, and the protection against tampering with the SSL handshake protocol relies heavily on the secrecy of the `master_secret`. Therefore, it

Figure 2: `master_secret` usage  
message usage

---

```
certificate verify  
    hash = adhoc-MAC(master_secret,  
                      handshake_messages)  
finished  
    hash = adhoc-MAC(master_secret,  
                      handshake_messages + sender)  
change cipher spec  
    key_block = expand-keys(  
        master_secret,  
        ServerHello.random +  
        ClientHello.random, ...)
```

is important that the `master_secret` be especially heavily guarded. In protocol design, this means that usage of the `master_secret` should be greatly limited.

Figure 2 lists all of the places where the `master_secret` is used. Each item in the list can be used to recover a relation involving the `master_secret` and some known plaintext.

An enemy can collect unlimited amounts of known plaintext for the `master_secret`-keyed MAC transformation found in the `finished` message. The informed adversary opens many simultaneous connections via `client hello` messages requesting the resumption of the targeted session. For each such connection, the server will pick a random nonce, calculate a MAC with the `master_secret`, and send it back encrypted in a `finished` message. The clever adversary should leave all those connections open without responding to the server's `finished` message: sending incorrect data on any of the connections will cause a fatal alert which makes the session unresumable. In this way, the opponent can collect great amounts of known plaintext hashed with the `master_secret`. If some cryptanalyst discovers an attack on `adhoc-MAC()` which uses much known plaintext to recover the secret key, the current SSL protocol could become unsafe. A strongly robust handshake protocol should probably limit the amount of known text that is available to a cryptanalyst.

The `pre_master_secret` is at least as important to protect. One way that an attacker may acquire more known text hashed with a `pre_master_secret` is to replay the original RSA-encrypted ciphertext which contained the `pre_master_secret`. The attacker will not be able to complete the SSL handshake protocol with this replayed RSA ciphertext, but it may be

possible to get the server to send a **finished** message containing some known plaintext hashed with the `pre_master_secret`. This will only be possible if the server is pipelined enough to send a **finished** message after receiving the **client key exchange** message but before receiving a client **finished** message. This trick would be impossible if the client's and server's random nonces were bound more tightly to the `pre_master_secret` in the RSA key-exchange—perhaps a hash of the nonces should be included in the RSA encryption input.

## 4.8 Diffie-Hellman key-exchange

SSL 3.0 includes support for ephemerally-keyed Diffie-Hellman key-exchange. Since Diffie-Hellman is the only public key algorithm known which can efficiently provide perfect forward secrecy, this is an excellent addition to SSL. In a SSL 3.0 Diffie-Hellman key-exchange, the server specifies its Diffie-Hellman exponent as well as the prime modulus and generator. To avoid server-generated trapdoors, the client should be careful to check that the modulus and generator are from a fixed public list of safe values. The well-known man-in-the-middle attack is prevented in SSL 3.0 by requiring the server's Diffie-Hellman exponential to be authenticated. (Anonymous clients are not required to possess a certificate.) There is no support for higher-performance variants of Diffie-Hellman, such as smaller (160-bit) exponents or elliptic curve variants.

## 4.9 The alert protocol

SSL includes a small provision for sending event-driven **alert** messages. Many of these indicate fatal error conditions and instruct the recipient to immediately tear down the session. For instance, the `close_notify` **alert** message indicates that the sender is finished sending application data on the connection; since **alert** messages are normally authenticated, this prevents a truncation attack. As another example, reception of any packet with an incorrect MAC will result in a fatal **alert**.

## 4.10 MAC usage

The SSL 3.0 handshake protocol uses several ad-hoc MAC constructions to provide message integrity. The security of these MACs has not been thoroughly evaluated. We believe that SSL 3.0 should consist-

ently use HMAC whenever a MAC is called for; ad-hoc MACs should be avoided.

## 4.11 Summary

The SSL handshake protocol has several vulnerabilities and worrisome features, especially in areas which have seen recent revision. These are only troublesome when active attacks are a concern. Furthermore, these are not universal weaknesses: different implementations may or may not be vulnerable. A flaw in a protocol does not necessarily yield a vulnerable implementation. Nonetheless, if the specification does not explicitly warn of an attack (or prevent it directly), it seems reasonable to offer constructive criticism.

## 5 Conclusion

This security analysis has dedicated the greatest amount of time to shortcomings of the SSL 3.0 protocol, but that was purely for reasons of exposition. One would be hard-pressed to find any correlation between the amount of space required to explain a technical point and its importance or severity. Therefore, it is worth putting the previous sections in perspective, reviewing the big picture, and summarizing the security of SSL 3.0.

In general SSL 3.0 provides excellent security against eavesdropping and other passive attacks. Although export-weakened modes offer only minimal confidentiality protection, there is nothing SSL can do to improve that fact. The only change to SSL's protection against passive attacks worth recommending is support for padding to stop traffic analysis of GET request lengths.

This analysis has revealed a number of active attacks on the SSL 3.0 protocol (though some implementations may not be vulnerable). The most important new attacks are `change cipher spec`-dropping, `KeyExchangeAlgorithm`-spoofing, and version rollback. The SSL specification should be changed to warn of these new attacks. Fortunately, it is not hard to patch up the small flaws which allowed these attacks, and several possible fixes were listed.

The analysis has also revealed several ways in which the robustness of the SSL protocol can be improved. Many remarks were not inspired by direct vulnerabilities, but still are worth considering for future versions of SSL. Many of the pitfalls in SSL 3.0 were

found in areas that have seen recent revision.

It is important not to overstate the practical significance of any of these flaws. Most of the weaknesses described in this note arise from a small oversight and can be corrected without overhauling the basic structure of the protocol. Of course, they are still worth fixing.

SSL 2.0 was subject to quite a number of active attacks on its record layer and key-exchange protocol. SSL 3.0 plugs those gaping holes and thus is considerably more secure against active attacks. SSL 3.0 also provides much better message integrity protection in export-weakened modes—the common case—than SSL 2.0 did: SSL 2.0 provided only 40-bit MACs in those modes, while SSL 3.0 always uses 128-bit MACs. Finally, SSL 3.0 improves a number of non-security aspects of SSL, such as flexible support for a wide variety of cryptographic algorithms. It seems fair to conclude that SSL 3.0 qualifies as a significant improvement over SSL 2.0.

In short, while there are still a few technical wrinkles to iron out, on the whole SSL 3.0 is a valuable step toward practical communications security for Internet applications.

## 6 Acknowledgements

We are indebted to Paul Kocher, who provided many valuable comments and corrections. Of course, any mistakes are solely our responsibility.

## References

- [BCK96] M. Bellare, R. Canetti, and H. Krawczyk, “Keying Hash Functions for Message Authentication,” *Advances in Cryptology—CRYPTO ’96 Proceedings*, Springer-Verlag, 1996, pp. 1–15.
- [Bel96] S. Bellovin, “Problem Areas for the IP Security Protocols”, *Proceedings of the Sixth USENIX Security Symposium*, Usenix Association, 1996, pp. 205–214.  
<ftp://ftp.research.att.com/dist/smb/badesp.ps>.
- [FKK96] A. Freier, P. Karlton, and P. Kocher, “The SSL Protocol Version 3.0”, <ftp://ftp.netscape.com/pub/review/ssl-spec.tar.Z>, March 4 1996, Internet Draft, work in progress.

- [Koc96] P. Kocher, personal communication, 1996.
- [KV83] V. Voydock and S. Kent, “Security Mechanisms in High-Level Network Protocols”, *ACM Computing Surveys*, v. 5, n. 2, June 1983, pp. 135–171.
- [PCT95] J. Benaloh, B. Lampson, D. Simon, T. Spies, and B. Yee, “Microsoft Corporation’s PCT Protocol”, October 1995, Internet Draft, work in progress.
- [RSA93] RSA Data Security, Inc., “Public-Key Cryptography Standards (PKCS),” Nov 93.
- [Sch96] B. Schneier, *Applied Cryptography, 2nd Edition*, John Wiley & Sons, 1996.
- [Yee96] B. Yee, personal communication, June 1996.

# Fast, Automatic Checking of Security Protocols

Darrell Kindred

*Computer Science Department*

*Carnegie Mellon University*

{dkindred,wing}@cs.cmu.edu

Jeannette M. Wing

## Abstract

Protocols in electronic commerce and other security-sensitive applications require careful reasoning to demonstrate their robustness against attacks. Several logics have been developed for doing this reasoning formally, but protocol designers usually do the proofs by hand, a process which is time-consuming and error-prone.

We present a new approach, *theory checking*, to analyzing and verifying properties of security protocols. In this approach we generate the entire finite theory,  $Th$ , of a logic for reasoning about a security protocol; determining whether it satisfies a property,  $\phi$ , is thus a simple membership test:  $\phi \in Th$ . Our approach relies on (1) modeling a finite instance of a protocol in the way that the security community naturally, though informally, presents a security protocol, and (2) placing restrictions on a logic's rules of inference to guarantee that our algorithm terminates, generating a finite theory. A novel benefit to our approach is that because of these restrictions we can provide an automatic *theory-checker generator*. We applied our approach and our theory-checker generator to three different logics for reasoning about authentication and electronic commerce protocols: the Burrows-Abadi-Needham logic of authentication, AUTLOG, and Kailar's accountability logic [4, 8, 6]. For each we verified the desired properties using specialized theory checkers; most checks took less than two minutes, and all less than fifteen.

## 1 Motivation for our Approach

Past approaches to reasoning about security protocols, e.g., authentication protocols, have relied on either pencil-and-paper proof or machine-assisted proof through the use of interactive theorem provers, e.g., the Boyer-Moore Prover [3], Gypsy [5], HDM [9], Ina

This research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. Views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of Wright Laboratory or the United States Government.

Jo [10], and UNISEX [7]. Proofs of properties of these protocols relied on either specialized logics, e.g., Burrows-Abadi-Needham's Logic of Authentication, or an encoding of a specialized logic in the theorem prover's general-purpose logic. These proofs are tedious to do. If done by hand, they are prone to error; and they can be applied only to small examples. If done by machine, they require tremendous user patience since the machine usually insists on treating the critically creative and the boring bookkeeping steps of the proof all with equal importance; they often take hours to complete, even discounting human user time; and they are also prone to error since they still rely on human intervention and ingenuity.

In this paper we present a completely different approach to verifying properties about security protocols. It relies on the observation that informal reasoning about these protocols by the security community is invariably done in terms of a *finite* number of entities, e.g., parties communicating, messages exchanged, types of messages, encryption and decryption keys. Thus, we reason about a finite model of a protocol rather than its generalization. Our approach further relies on the observation that logics used to reason about these protocols have a finite number of rules of inference that "grow in a controlled manner" (discussed in Section 2.1). It is this controlled growth that enables us to build an automatic checker for a given logic and class of protocols we wish to verify. It is the small size in the number of entities we need to model and the small size in the number of rules in any given logic that enables us to do this checking fast.

Stated simply, our method is to build the entire theory,  $Th$ , given a logic and a model of the protocol we want to check. Since the model is finite and since the logic's rules always shrink or grow in a controlled manner, the theory we generate is finite. Then checking whether a property,  $\phi$ , holds of a protocol boils down to a simple membership test:  $\phi \in Th$ ? In all the examples we have looked at, generating  $Th$  takes no more than a few minutes; membership is a trivial test.

In fact, we do even better. We provide more than a

single fast, automatic checker for verifying properties of a class of security protocols, but better yet, we provide a tool that for any given logic *generates* one of these fast, automatic checkers. In other words our tool is a *theory-checker generator* (and each checker it generates implements the method described above). And best of all, the tool we built generates these checkers completely automatically as well!

In this paper, we first describe our method and argue informally its correctness and termination properties. We illustrate in Section 3 how we applied this method to three different logics, each proposed as the basis for reasoning about security protocols:

Logic	Class of Protocols	Properties to Check
BAN	authentication	authentication
AUTLOG	authentication	authentication, privacy
Kailar	elec. commerce	accountability

We start with the BAN logic since AUTLOG is a variation of it and Kailar’s logic is similar in flavor to BAN. We discuss some of the implementation details in Section 4 and explain how we are able to build a tool that can generate checkers automatically. Section 5 discusses related work. We close in Section 6 by summarizing our contributions and discussing two important future directions of this work: why logics for reasoning about security protocols are particularly amenable to our method and the general applicability of our method to domains outside of security.

## 2 Our Method

The technique for checking whether some desired property,  $\phi$ , holds for a given protocol,  $P$ , consists of these basic steps:

- Given a logic,  $L$  (consisting of a finite set of axioms and controlled-growth rules), our tool automatically generates a checker,  $C$ , specialized to that logic.
- Given the protocol,  $P$ , for which we want to check property  $\phi$ , encode the initial assumptions and messages of  $P$  as formulas of  $L$ ; call this set  $T^0$ .
- Using  $C$ , exhaustively enumerate the theory,  $T^*$ , that is, the set of facts (formulas) derivable from the formulas in  $T^0$ .
- Determine whether  $\phi$  is in  $T^*$  by a simple membership test.

In the following sections, we explain the class of logics to which this method can be applied, give a more detailed description of the algorithm, and present informal arguments for its correctness and termination.

### 2.1 Class of Logics

The method we use requires several restrictions on the logic,  $L$ . First, formulas of the logic are the smallest set such that

- $V$  is a formula for any variable  $V$ , and
- if  $F_1, \dots, F_n$  are formulas and  $S$  is a function symbol, then  $S(F_1, \dots, F_n)$  is a formula (for any  $n \geq 0$ ).

A constant can be represented as a function symbol with no arguments (we will omit the parentheses in this case). In particular, no conjunction, disjunction, negation, or quantifiers are permitted. Here are some legal formulas:

```
believes(A, shared_key(Kab, A, B))
sees(B, encrypt(KBS, shared_key(Kab, A, B)))
controls(S, shared_key(Kab, A, B))
```

We must be able to separate the rules of inference of the logic  $L$  into three classes:

- *S-rules*: An S-rule (shrinking rule) consists of a set of premises,  $\{P_1, \dots, P_n\}$ , and a conclusion  $C$ , such that if  $P_1, \dots, P_n$  unify simultaneously with a set of valid<sup>1</sup> formulas, then  $C$  (appropriately instantiated) is a valid formula. The conclusion must be the same size as or smaller than one of the premises, by some well-founded measure. Each variable occurring in the conclusion must also occur in one or more premises.
- *G-rules*: A G-rule (growing rule) has the same form as an S-rule, but the conclusion must be strictly larger than each of the premises, by the same measure, and each variable occurring in the premises must also occur in the conclusion.
- *Rewrites*: A rewrite is a pair of formulas,  $(f_1, f_2)$ , such that any occurrence of  $f_1$  in a valid formula may be replaced by  $f_2$ , yielding another valid formula. The formula  $f_2$  must be the same size as  $f_1$  and contain the same variables.

The classification of a set of rules as S-rules and G-rules can be done automatically if an appropriate measure

<sup>1</sup>We mean *valid* in the technical sense; that is, derivable from the assumptions.

is supplied. The measure must be compositional; that is, replacing a subformula by a formula of the same measure must not change the measure of the whole formula. Typically, a simple measure such as the number of function symbols will suffice.

Finally, we require that each S-rule must still meet the S-rule criteria when all its premises that could match G-rule conclusions are removed. We will refer to this as the *S/G restriction* since it constrains the manner in which S- and G-rules can interact with each other. Whereas the other restrictions are local properties of individual rules and thus can be checked for each rule in isolation, this global restriction involves the whole set of rules.

## 2.2 The Algorithm

The encoding of a protocol in the target logic will normally take the form of a set of formulas that represent initial assumptions held by the involved parties and the effects of the messages sent during a run of the protocol. The core of the verification method is an algorithm for computing the transitive closure, under the rules of inference, of this initial set of valid formulas.

First, we consider the case in which there are S-rules, but no G-rules or rewrites. The basic strategy we use is a breadth-first traversal in which each node of the traversed dag is a valid formula. The roots of the dag, i.e., the initial fringe, consist of all the formulas representing assumptions and messages. At each step in the traversal, we consider all possible applications of the S-rules that *use* a formula from the fringe. By *use* we mean that at least one of the premises unifies with such a formula. The new fringe consists of all the new conclusions reached by those S-rule applications. By the argument given in Section 2.3, this process will eventually halt. The resulting set of formulas is the complete set of valid formulas.

Next, we introduce the G-rules. The G-rules must be applied more judiciously than the S-rules, since applying them eagerly can produce infinitely many valid formulas. We apply them only as necessary to enable further application of S-rules. For each S-rule, we separate its premises into those that unify with some G-rule conclusion and those that do not. When applying that S-rule, we first unify the latter group of premises with known valid formulas. We can then proceed to search for a match for the remaining premises by applying the G-rules “in reverse” to the premises until we either reach a set of formulas all of which are known valid, or we cannot apply the G-rules further. (This process is guaranteed to terminate: see Section 2.3.)

Finally, we introduce the rewrites. Since they neither produce larger formulas nor introduce new variables, we

could apply them eagerly like the S-rules. This can lead to exponential blowup in the set of valid formulas, though, so it is better to use them only as needed. Since rewrites can be applied to any subformula, we augment a simple unification algorithm by trying rewrites at each recursive step of the unification [15]. Plotkin described a similar technique for building equational axioms into the unification process [16].

Since the G-rules and rewrites are applied lazily, the full algorithm does *not* generate the complete set of valid formulas. Any formula whose formal proof has a rewrite or G-rule as its last step may not appear. The generated set of formulas will have the property that any valid formula will be reachable from this set by a proof involving only G-rules and rewrites. In practice, we have found that the G-rules are often used solely for establishing side-conditions, and that all the “interesting” formulas will in fact appear in the generated set (possibly after applying rewrites). Nonetheless, if we are interested in a specific formula we can easily test whether it holds by performing a simple preprocessing step prior to testing for membership in the generated set.

## 2.3 Correctness and Termination

The soundness of the algorithm is easy to establish. For any formula that is added to the fringe, we can easily construct a proof of it by tracing back up the dag to formulas in the initial valid set.

To show completeness (i.e., that any valid formula is reachable from the generated set using only G-rules and rewrites), we use induction on the number of S-rule applications in the proof of a formula,  $F$ . If there exists a proof using no S-rule applications, then the initial valid set of formulas is sufficient, since we only need to capture a set of formulas from which  $F$  can be reached by G-rules and rewrites. If there exists a proof of  $F$  using  $n$  S-rule applications, we know by the induction hypothesis that all lines in the proof before the  $n$ th S-rule application are either in the valid set or can be reached from an element of the valid set by applying G-rules and rewrites. Since we apply as many G-rules and rewrites as necessary to enable S-rule applications, the result of the  $n$ th S-rule application will be in the valid set, and thus  $F$  is reachable from the valid set using only G-rules and rewrites.

The algorithm is guaranteed to terminate because of the restrictions we impose on the rules. Consider a modified logic in which any S-rule premise that could match a G-rule conclusion is removed, and the G-rules are eliminated (since they cannot now make any contribution). The modified S-rules still meet the S-rule criteria by the S/G restriction.

If we run the algorithm on this modified logic with some finite set of initial assumptions, each new formula is added as the result of an S-rule and thus is smaller or the same size as one of its premises. The rewrites can only be applied finitely many times since they preserve the formula's size and set of variables. Thus we will never produce a formula that is larger than the largest initial assumption, and we will introduce no new variables, so the set of formulas is finite and the algorithm must terminate.

If we then reintroduce the G-rules and the missing premises and run the algorithm, every S-rule application will correspond to exactly one S-rule application from the first run. Furthermore, each reverse-application of the G-rules will terminate since G-rules are strictly growing, so the algorithm terminates.

### 3 Examples

We used our theory-checker generator to build three theory checkers, encoding three different logics: the well-known BAN logic of authentication [4]; AUTLOG [8], an extension of the BAN logic; and Kailar's logic for reasoning about accountability in electronic commerce protocols [6]. We describe each of these encodings below.

#### 3.1 BAN

The BAN logic is a natural case to consider; it motivated the original development of our method and tool. This logic is normally applied to authentication protocols. It allows certain notions of belief and trust to be expressed in a simple manner, and it provides rules for interpreting encrypted messages exchanged among the parties (principals) involved in a protocol.

##### 3.1.1 The Logic

In encoding the BAN logic and its accompanying sample protocols, we had to make several adjustments and additions to the logic as originally presented [4], to account for rules and assumptions that were missing or implicit.

Figure 1 shows the functions used in the encoding. The first eleven correspond directly to constructs in the original logic, and have clear interpretations. The last two were added: *inv* makes explicit the relationship implied between  $K$  and  $K^{-1}$ , and *distinct* expresses that two principals are different.

The BAN logic consists of eleven basic rules of inference:

- three message-meaning rules that allow one principal to deduce that a given message was once uttered

by some other principal;

- one nonce-verification rule whereby a principal can determine that some message was sent recently, by examining nonces;
- one jurisdiction rule, expressing one principal's trust of another;
- five rules for extracting components of messages, some requiring knowledge of the appropriate keys; and
- one freshness rule, which states that a conjunction is fresh if any part of it is fresh.

We encode each of these rules directly as a single S-rule, with the exception of the freshness rule, which is a G-rule since its conclusion is larger than its premise:

$$\frac{\text{believes}(P, \text{fresh}(X))}{\text{believes}(P, \text{fresh}(\text{comma}(X, Y)))}$$

The message-meaning and extraction rules involving encryption carry a side-condition that the principal who encrypted the message is different from the one interpreting it. We encode this explicitly using ■ three-place *encrypt* function and the extra *distinct* function, by adding an extra premise to each of these rules.

We add a few rules to this original set to make it more complete. There are two additional component-extraction S-rules:

$$\frac{\text{believes}(P, \text{said}(Q, \text{comma}(X, Y)))}{\text{believes}(P, \text{said}(Q, X))}$$

$$\frac{\text{believes}(P, \text{believes}(Q, \text{comma}(X, Y)))}{\text{believes}(P, \text{believes}(Q, X))}$$

The BAN Kerberos protocol analysis indirectly refers to (and depends on) these rules; AUTLOG includes them explicitly.

We also add two S-rules that do the work of message-meaning and nonce-verification simultaneously. The shared-key version of this rule is

$$\frac{\begin{array}{l} \text{believes}(P, \text{fresh}(K)) \\ \text{believes}(P, \text{shared\_key}(K, Q, P)) \\ \text{sees}(P, \text{encrypt}(X, K, R)) \\ \text{distinct}(P, R) \end{array}}{\text{believes}(P, \text{believes}(Q, X))}$$

One of these rules is implicitly required by the BAN Andrew secure RPC analysis. AUTLOG handles this case somewhat more elegantly by introducing a “recently said” notion, adding extra conclusions to its message-meaning (“authentication”) rules, and creating new “contents” rules.

Function	BAN interpretation
<i>believes</i> ( $P, X$ )	$P$ believes $X$
<i>sees</i> ( $P, X$ )	$P$ sees $X$
<i>said</i> ( $P, X$ )	$P$ said $X$
<i>controls</i> ( $P, X$ )	$P$ controls $X$
<i>fresh</i> ( $X$ )	$\text{fresh}(X)$
<i>shared_key</i> ( $K, P, Q$ )	$P \xleftarrow{K} Q$
<i>public_key</i> ( $K, P$ )	$\xleftarrow{K} P$
<i>secret</i> ( $Y, P, Q$ )	$P \xrightarrow{K} Q$
<i>encrypt</i> ( $X, K, P$ )	$\{X\}_K$ from $P$
<i>combine</i> ( $X, Y$ )	$\langle X \rangle_Y$
<i>comma</i> ( $X, Y$ )	$X, Y$ (conjunction)
<i>inv</i> ( $K_1, K_2$ )	$K_1$ and $K_2$ are a public/private key pair
<i>distinct</i> ( $P, Q$ )	principals $P$ and $Q$ are not the same

Figure 1: BAN functions

We add seven freshness G-rules: four to reflect the fact that an encrypted (or combined) message is fresh if either the body or the key is, and three that extend the freshness of a key to freshness of statements about that key. The example protocol verifications in the BAN paper require some of these extra freshness rules, and we include the rest for completeness. AUTLOG includes the first four of them.

Finally, since we represent message composition explicitly (via *comma*), we include three rewrites that express the commutativity and associativity of the *comma* function; three more rewrites provide commutativity for *shared\_key*, *secret*, and *distinct*. The *shared\_key* rewrite looks like this:

$$\frac{\text{shared\_key}(K, P, Q)}{\text{shared\_key}(K, Q, P)}$$

Appendix A contains the complete set of rules and rewrites.

The logic restrictions from Section 2 do not permit the use of universal quantifiers, as BAN specifications sometimes do in delegation statements [4]:

$$A \text{ believes } \forall K. (S \text{ controls } (A \xleftarrow{K} B))$$

However, since none of the BAN rules introduce new keys, we can get the effect of this universal quantification in assumptions by instantiating the statement with each of the keys mentioned in the other assumptions. We could do this automatically as a preprocessing step. It may be

possible to extend our method slightly to allow universal quantification at the outermost level.

After encoding the rules, we entered each of the four protocols examined in the BAN paper and checked all the properties claimed there [4]. (We added most of the extensions above after some verification attempt failed.)

### 3.1.2 Kerberos

Through a sequence of four messages, the Kerberos protocol establishes a shared key for communication between two principals, using a trusted server [12]. The BAN analysis of this protocol starts by constructing a three-message idealized protocol; the idealized protocol ignores message 1 since it is unencrypted. The BAN analysis then goes on to list ten initial assumptions regarding client/server shared keys, trust of the server, and freshness of the timestamps used [4]. We express each of these three messages and ten assumptions directly (the conversion is purely syntactic), and add four more assumptions (see Figures 2 and 3).

The first extra assumption—that  $A$  must believe its own timestamp to be fresh—is missing in [4], and the last three are required to satisfy the distinctness side-conditions. After making these adjustments, we can run the 14 initial assumptions and 3 messages through our automatically generated BAN-checker, and it will generate an additional 50 true formulas in 90 seconds<sup>2</sup>.

<sup>2</sup>All timings were done on an IBM RS/6000 model 25T with an 80MHz PowerPC 601 CPU.

---

**Message 2.**  $S \rightarrow A : \{T_s, A \xleftarrow{K_{ab}} B, \{T_s, A \xleftarrow{K_{ab}} B\}_{K_{bs}}\}_{K_{as}}$   
 $\quad sees(A, encrypt(comma(comma(T_s, shared\_key(K_{ab}, A, B)),$   
 $\quad \quad encrypt(comma(T_s, shared\_key(K_{ab}, A, B)),$   
 $\quad \quad \quad K_{bs}, S))),$   
 $\quad \quad K_{as}, S))$

**Message 3.**  $A \rightarrow B : \{T_s, A \xleftarrow{K_{ab}} B\}_{K_{bs}}, \{T_a, A \xleftarrow{K_{ab}} B\}_{K_{ab}} from A$   
 $\quad sees(B, comma(encrypt(comma(T_s, shared\_key(K_{ab}, A, B)), K_{bs}, S),$   
 $\quad \quad encrypt(comma(T_a, shared\_key(K_{ab}, A, B)), K_{ab}, A)))$

**Message 4.**  $B \rightarrow A : \{T_a, A \xleftarrow{K_{ab}} B\}_{K_{ab}} from B$   
 $\quad sees(A, encrypt(comma(T_a, shared\_key(K_{ab}, A, B)), K_{ab}, B))$

Figure 2: Kerberos protocol messages, in BAN idealized form and converted to our syntax.

---

$believes(A, shared\_key(K_{as}, S, A))$   
 $believes(B, shared\_key(K_{bs}, S, B))$   
 $believes(S, shared\_key(K_{as}, A, S))$   
 $believes(S, shared\_key(K_{bs}, B, S))$   
 $believes(S, shared\_key(K_{ab}, A, B))$   
 $believes(A, controls(S, shared\_key(K_{ab}, A, B)))$   
 $believes(B, controls(S, shared\_key(K_{ab}, A, B)))$   
 $believes(A, fresh(T_s))$   
 $believes(B, fresh(T_s))$   
 $believes(B, fresh(T_a))$   
  
 $believes(A, fresh(T_a))$   
 $distinct(A, S)$   
 $distinct(A, B)$   
 $distinct(B, S)$

Figure 3: Encoding of the Kerberos initial assumptions. All but the last four assumptions appear in the BAN analysis [4].

---

By running a simple membership test, we verified that these four desired results are among them:

$$\begin{aligned} &\text{believes}(A, \text{shared\_key}(K_{ab}, A, B)) \\ &\text{believes}(B, \text{shared\_key}(K_{ab}, A, B)) \\ &\text{believes}(B, \text{believes}(A, \text{shared\_key}(K_{ab}, A, B))) \\ &\text{believes}(A, \text{believes}(B, \text{shared\_key}(K_{ab}, A, B))) \end{aligned}$$

These results agree with the original BAN analysis. They indicate that each of the two parties believes it shares a key with the other, and that each believes that the other believes the same thing. Appendix B illustrates one step in the algorithm: applying an S-rule with the help of a G-rule and rewrites.

If we remove the optional final message from the protocol and run the checker again, it will generate 41 valid formulas. By computing the difference between this set and the first set of 50, we can determine exactly what the final message contributes. Among the 9 formulas in this difference is

$$\text{believes}(A, \text{believes}(B, \text{shared\_key}(K_{ab}, A, B))),$$

(the last of the four results above). This confirms the claim in the original analysis that “the three-message protocol does not convince  $A$  of  $B$ ’s existence” [4]. This technique of examining the set difference between the deduced properties of two versions of a protocol is a simple but powerful benefit of our approach; it helps in understanding differences between protocol variants and it supports “rapid” protocol design.

### 3.1.3 Andrew RPC, Needham-Schroeder, and CCITT X.509

We encoded the assumptions and messages of the three variants of the Andrew secure RPC protocol given in the BAN paper, and got the expected results. The last of these verifications required an extra freshness assumption not mentioned in the BAN analysis, as well as some of our added freshness rules and the simultaneous message-meaning/nonce-verification rule.

We duplicated the BAN results for two variants of the Needham-Schroeder public-key secret-exchange protocol. Finally, we ran the checker on two variants of the CCITT X.509 protocol explored in the BAN paper. One of these checks failed to produce the expected results, and this led us to discover an oversight in the BAN analysis: they observe a weakness in the original X.509 protocol and claim, “The simplest fix is to sign the secret data  $Y_a$  and  $Y_b$  before it is encrypted for privacy.” In fact we must sign the secret data together with a nonce to ensure freshness. After we corrected this, the verifications proceeded as expected.

## 3.2 AUTLOG

AUTLOG is an extension of the BAN logic, proposed by Kessler and Wedel [8]. It introduces several concepts, including a simulated eavesdropper for detecting information leaks, and the idea of principals “recognizing” decrypted messages.

Our encoding of AUTLOG uses all the BAN functions, and a few extras: *recognizable*, *mac* (for “message authentication codes”), *hash*, and *recently\_said*. The original rules of inference from AUTLOG can be entered almost verbatim. There are 23 S-rules and 19 G-rules; the rules governing freshness and recognition are the only G-rules.

To check ■ protocol for leaks using AUTLOG, one finds the closure over the “seeing” rules of the transmitted messages. The resulting list will include everything an eavesdropper could see. Our tool is well-suited to computing this list; the seeing rules are all S-rules, so the checker will generate exactly the desired list.

Kessler and Wedel present two simple challenge-response protocols: one in which only the challenge is encrypted and another in which only the response is encrypted. We encoded both of these protocols and verified the properties Kessler and Wedel claim: that both achieve the authentication goal

$$\text{believes}(B, \text{recently\_said}(A, R_B))$$

where  $R_B$  is the secret  $A$  provides to prove its identity. Furthermore, through the eavesdropper analysis mentioned above, we can show that in the encrypted-challenge version, the secret is revealed and thus the protocol is insecure. (The BAN logic cannot express this.)

We also checked that the Kerberos protocol, expressed in AUTLOG, satisfied properties similar to those described in Section 3.1. Since AUTLOG has a large set of rules, this verification took roughly 13 minutes; this was significantly longer than any other verification. In Section 4 we mention some optimizations we expect to reduce this time.

## 3.3 Kailar’s Accountability Logic

More recently, Kailar has proposed a simple logic for reasoning about accountability in electronic commerce protocols [6]. The central construct in this logic is

$$P \text{ CanProve } X$$

which means that principal  $P$  can convince anyone in an intended audience sharing a set of assumptions that  $X$  holds, without revealing any “secrets” other than  $X$ .

We encoded the version of this logic using “strong proof” and “global trust”, but the weak-proof and

nonglobal-trust versions would be equally simple. We used these functions: *CanProve*, *IsTrustedOn*, *Implies*, *Authenticates*, *Says*, *Receives*, *SignedWith*, *comma*, and *inv*.

We encoded the four main rules of the logic as follows:

$$\begin{array}{l} \text{Conj : } \frac{\text{CanProve}(P, X), \text{CanProve}(P, Y)}{\text{CanProve}(P, \text{comma}(X, Y))} \\ \\ \text{Inf : } \frac{\text{CanProve}(P, X), \text{Implies}(X, Y)}{\text{CanProve}(P, Y)} \\ \\ \text{Sign : } \frac{\text{Receives}(P, \text{SignedWith}(M, K^{-1}))}{\text{CanProve}(P, \text{Authenticates}(K, Q))} \\ \qquad \qquad \qquad \text{Inv}(K, K^{-1}) \\ \qquad \qquad \qquad \text{CanProve}(P, \text{Says}(Q, M)) \\ \\ \text{Trust : } \frac{\text{CanProve}(P, \text{Says}(Q, X))}{\text{CanProve}(P, \text{IsTrustedOn}(Q, X))} \\ \qquad \qquad \qquad \text{CanProve}(P, X) \end{array}$$

The **Conj** and **Inf** rules allow building conjunctions and using initially-assumed implications. The **Sign** and **Trust** rules correspond roughly to the BAN logic's public-key message-meaning and jurisdiction rules. We replace the construct *X in M* (representing interpretation of part of a message) with three explicit rules for extracting components of a message. We add rewrites expressing the commutativity and associativity of *comma*, as in the other logics. This encoding does not require any G-rules.

We verified the variants of the IBS (NetBill) electronic payment protocol that Kailar analyzes [6]. Figure 4 contains an encoding of part of the “service provision” phase of the asymmetric-key version of this protocol. If we run the Kailar-logic checker on these messages and assumptions, it will apply the **Sign** rule to produce these two formulas:

$$\begin{aligned} & \text{CanProve}(S, \text{Says}(E, \text{comma}(\text{SignedWith}(\text{Price}, \\ & \qquad \qquad \qquad K_s^{-1}), \\ & \qquad \qquad \qquad \text{Price}))), \\ & \text{CanProve}(S, \text{Says}(E, \text{ServiceAck})). \end{aligned}$$

It will then apply the component-extracting rule to produce

$$\begin{aligned} & \text{CanProve}(S, \text{Says}(E, \text{SignedWith}(\text{Price}, K_s^{-1}))) \\ & \text{CanProve}(S, \text{Says}(E, \text{Price})). \end{aligned}$$

Finally, it will apply **Inf** to derive these results, which Kailar gives [6].

$$\begin{aligned} & \text{CanProve}(S, \text{Says}(E, \text{ReceivedOneServiceItem}(E))) \\ & \text{CanProve}(S, \text{Says}(E, \text{AgreesToPrice}(E, pr))) \end{aligned}$$

It will stop at this point, since no further rule applications can produce new formulas.

We verified the rest of Kailar’s results for two variants of the IBS protocol and for the SPX Authentication Exchange protocol. Each check with this logic took less than ten seconds.

## 4 Implementation

Implemented in the Standard ML programming language, our tool makes heavy use of SML’s *modules* support. SML modules can be *signatures* (interface descriptions), *structures* (implementations), or *functors* (generic or parameterized implementations). With our tool, each logic is represented by a structure containing its function names (e.g., **believes** and **sees**), S-rules, G-rules, rewrites, and a measure function on its formulas. The checker-generator is a functor that takes as an argument a logic module, and generates a new module which is a checker specialized to that logic. For instance, to create the three checkers we used, we invoke the functor, *TheoryChecker*, with three different Logic structures:

```
structure BanChecker =
  TheoryChecker(structure Logic = BAN
                ...)

structure AutlogChecker =
  TheoryChecker(structure Logic = AUTLOG
                ...)

structure KailarChecker =
  TheoryChecker(structure Logic = Kailar
                ...)
```

Then, for a particular protocol such as Kerberos, we first compute the closure of the list of initial assumptions appended (@) with the list of messages, and then we run a simple check for any desired property:

```
val formulas = BanChecker.closure
  (Kerberos.assumptions
   @ Kerberos.messages);
check(formulas, Kerberos.A_knows_Kab);
```

The current implementation is a rough first cut. We plan to make it significantly more efficient by representing sets of formulas with more sophisticated data structures that support fast lookups, union, and fast unification against the whole set of formulas; we also know of some possible optimizations to the search procedure which should help. Nonetheless, all but three of our verification attempts ran in less than two minutes each, and the remaining three took less than fifteen minutes each.

One way in which the current implementation could be improved is to allow type declarations for the functions in each logic. For instance, the BAN logic might

---

IBS protocol messages:

Message 3.  $E \rightarrow S : \{\{Price\}_{K_s^{-1}}, Price\}_{K_e^{-1}}$   
 $Receives(S, SignedWith(comma(SignedWith(Price, K_s^{-1}),$   
 $Price),$   
 $K_e^{-1}))$

Message 5.  $S \rightarrow E : \{Service\}_{K_s^{-1}}$   
 $Receives(E, SignedWith(Service, K_s^{-1}))$

Message 6.  $E \rightarrow S : \{ServiceAck\}_{K_e^{-1}}$   
 $Receives(S, SignedWith(ServiceAck, K_e^{-1}))$

Initial assumptions:

$CanProve(S, Authenticates(K_e, E))$   
 $Implies(Says(E, Price), AgreesToPrice(E, pr))$   
 $Implies(Says(E, ServiceAck), ReceivedOneServiceItem(E))$

Figure 4: Excerpt from IBS protocol and initial assumptions.

---

have types representing principals, keys, and formulas, and the *encrypt* function would expect a formula, a key, and a principal as its arguments. This would help eliminate errors in encoding both protocols and rules, and the search mechanism can then take advantage of the type information as well.

## 5 Related Work

Model checking is a technique whereby a finite state machine is checked for a property through exhaustive case analysis. It has been used successfully in hardware verification and protocol analysis. Most recently, the FDR model checker [17] has been used by Lowe [11] to debug and fix the Needham-Schroeder public key protocol and by Roscoe [18] to check noninterference of a simple security hierarchy (high/low). Like model checking, our method relies on the finiteness of the entity being verified; unlike model checking, however, we generate a finite theory, not a finite model.

Theorem proving, usually machine-assisted, is the more traditional approach to verifying security properties, including a long history of work based on the Bell-LaPadula model [1]. As in theorem proving, we manipulate the syntactic representation, i.e., the logic, of the entity we are verifying, by restricting the nature of the logic, however, unlike machine-assisted theorem proving,

we enumerate the entire theory rather than (with human assistance) develop lemmas and theorems as needed. We also express the messages exchanged in ■ protocol as a set of initial non-logical axioms (and thus express them in the same language as the logic), avoiding the need for the user to learn more than one input language. Moreover, our method is completely automatic and fast.

Before building our theory-checker generator, we implemented the BAN logic within the PVS verification system [14], and reproduced the Kerberos protocol proofs with it. The encoding of BAN in PVS was quite natural, but the proofs were tedious, primarily because for each rule application, we had to enter the variable instantiations manually since the prover could not guess the right ones. This would be a smaller problem in a verification system with support for unification.

Our approach is the closest in nature to logic programming, embodied in programming languages like Prolog. Indeed, AUTLOG has been implemented in Prolog. Our approach, however, is more powerful because we can produce results without any dependence on rule ordering, and because we use ■ modified unification algorithm that can handle simple rewrite rules. At the same time, we are more specific; by tailoring our reasoning to security protocols, we exploit the nature of the domain in ways that make exhaustive enumeration a computationally feasible

and tractable approach to automatic verification.

The idea of computing the “closure” (or “completion”) of a theory is used in the theorem proving system SATURATE [13]. Our restrictions on the input logic allow us to generate a saturated set of formulas that we find easier to interpret than the sets generated by these more general systems.

Finally, our approach makes it easy to generate specialized checkers automatically. Just as Jon Bentley has argued the need for “little languages” [2], our tool provides a way to construct “little tools” for “little logics.”

## 6 Summary and Future Directions

Our approach was motivated by the need to debug protocols in the security domain. When someone presents a security protocol, there is always an uneasiness on our part. These are typical questions that we pose ourselves when simply trying to understand a security protocol:

- Why is that message needed?
- Is the ordering of these messages required or incidental?
- Why is that message or part of the message encrypted? Is it necessary?

To argue the correctness, or better yet to find performance optimization, for example by deleting a message, unordering messages (and thus allowing concurrency), or avoiding encryption, necessitates at least a systematic way of reasoning about these protocols. Using specialized logics helps; using tools that automate these logics helps even more.

The restrictions we impose on the input logic offer the substantial advantage that we can guarantee termination, and furthermore a logic can be automatically checked for compliance with these restrictions (given a measure).

A further advantage of our approach, as illustrated by the “diff” example of Section 3.1.2, is that we can study closely-related theories by examining the formulas that appear in the generated set of one but not the other.

With fast, automatic verification, through the use of our checkers for “little logics,” protocol designers can invent and debug their protocols quickly, with the same ease as compiling a program. Thus as a compiler gives programmers assurance that their programs are type correct, our checkers can give protocol designers additional assurance that their protocols are “correct” (i.e., satisfy certain desired properties).

With the explosion of the Internet and the wide range of electronic commerce protocols proposed and in commercial use, the problem of verifying and debugging these protocols is going to get worse. Even well-understood authentication protocols are subject to attacks by the environment that do not satisfy their original assumptions.

Our method has two promising future directions. First, we readily acknowledge that we greatly exploit the nature of the domain: security protocols are most often explained informally in terms of a small number of parties (Alice, Bob, a server, and perhaps an eavesdropper), a small number of message exchanges (usually not more than ten), a small number of keys (public and private keys for each party involved), a small number of nested encryptions (usually under two), and so on. We also exploit the smallness of the logics involved: the BAN logic (as we encode it) has only twenty rules of inference; moreover, it has only eight G-rules. It is possible (though we have only an intuition at this point) that there is something inherent to security protocols and logics for reasoning about them that makes our theory-generation technique especially appropriate.

Second, of course, our technique is not only applicable to protocols in security. While we do impose significant restrictions on the logics, these restrictions are expressed in general terms and may well be satisfied by useful logics from other domains.

## A BAN logic encoding

This appendix contains the complete encoding of the BAN logic, from which the BAN checker is automatically generated. See Section 3.1 for further explanation.

The three message-meaning S-rules:

$$\begin{array}{c} \text{believes}(P, \text{shared\_key}(K, Q, P)) \\ \text{sees}(P, \text{encrypt}(X, K, R)) \\ \text{distinct}(P, R) \\ \hline \text{believes}(P, \text{said}(Q, X)) \end{array}$$

$$\begin{array}{c} \text{believes}(P, \text{public\_key}(K_1, Q)) \\ \text{sees}(P, \text{encrypt}(X, K_2, R)) \\ \text{inv}(K_1, K_2) \\ \text{distinct}(P, R) \\ \hline \text{believes}(P, \text{said}(Q, X)) \end{array}$$

$$\begin{array}{c} \text{believes}(P, \text{secret}(Y, Q, P)), \text{ sees}(P, \text{combine}(X, Y)) \\ \hline \text{believes}(P, \text{said}(Q, X)) \end{array}$$

The nonce-verification S-rule:

$$\begin{array}{c} \text{believes}(P, \text{said}(Q, X)) \\ \text{believes}(P, \text{fresh}(X)) \\ \hline \text{believes}(P, \text{believes}(Q, X)) \end{array}$$

The jurisdiction S-rule:

$$\frac{\text{believes}(P, \text{controls}(Q, X)) \\ \text{believes}(P, \text{believes}(Q, X))}{\text{believes}(P, X)}$$

The seven S-rules for extracting components of messages:

$$\begin{aligned} & \frac{\text{believes}(P, \text{shared\_key}(K, Q, P)) \\ \text{sees}(P, \text{encrypt}(X, K, R)) \\ \text{distinct}(P, R)}{\text{sees}(P, X)} \\ \\ & \frac{\text{believes}(P, \text{public\_key}(K, P)) \\ \text{sees}(P, \text{encrypt}(X, K, R))}{\text{sees}(P, X)} \\ \\ & \frac{\text{believes}(P, \text{public\_key}(K_1, Q)) \\ \text{sees}(P, \text{encrypt}(X, K_2, R)) \\ \text{inv}(K_1, K_2) \\ \text{distinct}(P, R)}{\text{sees}(P, X)} \\ \\ & \frac{\text{sees}(P, \text{combine}(X, Y))}{\text{sees}(P, X)} \\ \\ & \frac{\text{believes}(P, \text{comma}(X, Y))}{\text{believes}(P, X)} \\ \\ & \frac{\text{sees}(P, \text{comma}(X, Y))}{\text{sees}(P, X)} \\ \\ & \frac{\text{believes}(P, \text{said}(Q, \text{comma}(X, Y)))}{\text{believes}(P, \text{said}(Q, X))} \\ \\ & \frac{\text{believes}(P, \text{believes}(Q, \text{comma}(X, Y)))}{\text{believes}(P, \text{believes}(Q, X))} \end{aligned}$$

The two combined message-meaning and nonce-verification S-rules:

$$\begin{aligned} & \frac{\text{believes}(P, \text{fresh}(K)) \\ \text{sees}(P, \text{encrypt}(X, K, R)) \\ \text{distinct}(P, R)}{\text{believes}(P, \text{shared\_key}(K, Q, P))} \\ \\ & \frac{\text{believes}(P, \text{fresh}(Y)) \\ \text{sees}(P, \text{combine}(X, Y)) \\ \text{believes}(P, \text{secret}(Y, Q, P))}{\text{believes}(P, \text{believes}(Q, X))} \end{aligned}$$

The eight G-rules dealing with freshness:

$$\frac{\text{believes}(P, \text{fresh}(X))}{\text{believes}(P, \text{fresh}(\text{comma}(X, Y)))}$$

$$\begin{aligned} & \frac{\text{believes}(P, \text{fresh}(K))}{\text{believes}(P, \text{fresh}(\text{shared\_key}(K, Q, R)))} \\ \\ & \frac{\text{believes}(P, \text{fresh}(K))}{\text{believes}(P, \text{fresh}(\text{public\_key}(K, Q)))} \\ \\ & \frac{\text{believes}(P, \text{fresh}(Y))}{\text{believes}(P, \text{fresh}(\text{secret}(Y, Q, R)))} \\ \\ & \frac{\text{believes}(P, \text{fresh}(Y))}{\text{believes}(P, \text{fresh}(\text{combine}(X, Y)))} \\ \\ & \frac{\text{believes}(P, \text{fresh}(K))}{\text{believes}(P, \text{fresh}(\text{encrypt}(X, K, R)))} \\ \\ & \frac{\text{believes}(P, \text{fresh}(X))}{\text{believes}(P, \text{fresh}(\text{encrypt}(X, K, R)))} \\ \\ & \frac{\text{believes}(P, \text{fresh}(X))}{\text{believes}(P, \text{fresh}(\text{combine}(X, Y)))} \end{aligned}$$

Finally, the various commutativity and associativity rewrites:

$$\begin{aligned} & \frac{\text{comma}(X, Y)}{\text{comma}(Y, X)} \\ \\ & \frac{\text{comma}(\text{comma}(X, Y), Z)}{\text{comma}(X, \text{comma}(Y, Z))} \\ \\ & \frac{\text{comma}(X, \text{comma}(Y, Z))}{\text{comma}(\text{comma}(X, Y), Z)} \\ \\ & \frac{\text{believes}(P, \text{shared\_key}(K, Q, R))}{\text{believes}(P, \text{shared\_key}(K, R, Q))} \\ \\ & \frac{\text{believes}(P, \text{secret}(Y, Q, R))}{\text{believes}(P, \text{secret}(Y, R, Q))} \\ \\ & \frac{\text{distinct}(P, Q)}{\text{distinct}(Q, P)} \end{aligned}$$

## BAN checker sample step

Here we illustrate one step in the algorithm as applied to the BAN/Kerberos example from Section 3.1.2, to show how a new formula gets added to the fringe. After two levels of the breadth-first traversal are completed, there are 37 formulas in the known-valid set. Of these, 16 are in the fringe, including this one:

$$\text{believes}(B, \text{said}(S, \text{comma}(T_S, \text{shared\_key}(K_{ab}, A, B))))$$

This formula unifies with the first premise of the “nonce-verification” S-rule (see Appendix A), so we apply its unifier to the second premise of that rule, yielding

$$\text{believes}(B, \text{fresh}(\text{comma}(T_S, \text{shared\_key}(K_{ab}, A, B)))) .$$

None of the 37 formulas unifies with this additional premise, so we attempt to work backwards from it, using G-rules and rewrites. If we reverse-apply the first freshness G-rule, we get

$$\text{believes}(B, \text{fresh}(T_S)),$$

which is one of the initial assumptions of the protocol (and thus one of the 37 known formulas). Since all premises for the nonce-verification rule have now been matched, we insert its (instantiated) conclusion into the new fringe:

$$\begin{aligned} \text{believes}(B, \text{believes}(S, \text{comma}(T_S, \\ \text{shared\_key}(K_{ab}, \\ A, B))))). \end{aligned}$$

## References

- [1] D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, The MITRE Corporation, Bedford, MA, March 1976.
- [2] J. Bentley. Little languages. *Communications of the ACM*, 29(8):711–721, 1986.
- [3] R. S. Boyer and J. S. Moore. *A Computational Logic*. ACM monograph series. Academic Press, New York, 1979.
- [4] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
- [5] D. I. Good, R. L. London, and W. W. Bledsoe. An interactive program verification system. *IEEE Transactions on Software Engineering*, 1(1):59–67, 1979.
- [6] Rajashekhar Kailar. Accountability in electronic commerce protocols. *IEEE Transactions on Software Engineering*, 22(5):313–328, May 1996.
- [7] R. A. Kemmerer and S. T. Eckmann. *A User's Manual for the UNISEX System*. Dept. of Computer Science, UCSB, 1983.
- [8] Volker Kessler and Gabriele Wedel. AUTLOG—an advanced logic of authentication. In *Proc. the Computer Security Foundations Workshop VII*, pages 90–99. IEEE Comput. Soc., June 1994.
- [9] K. N. Levitt, L. Robinson, and B. A. Silverberg. The HDM handbook, vols. 1–3. Technical report, SRI International, Menlo Park, California, 1979.
- [10] R. Locasso, J. Scheid, D. V. Schorre, and P. R. Egger. The Ina Jo reference manual. Technical Report TM-(L)-6021/001/000, System Development Corporation, Santa Monica, California, 1980.
- [11] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055. Springer-Verlag, March 1996. Lecture Notes in Computer Science.
- [12] S. P. Miller, C. Neuman, J. I. Schiller, and J. H. Saltzer. *Kerberos authentication and authorization system*, chapter Sect. E.2.1. MIT, Cambridge, Massachusetts, July 1987.
- [13] P. Nivela and R. Nieuwenhuis. Saturation of first-order (constrained) clauses with the Saturate system. In *Proc. of the Fifth International Conference on Rewriting Techniques and Applications*, pages 436–440, June 1993.
- [14] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.
- [15] Laurence C. Paulson. *ML for the Working Programmer*. Cambridge University Press, Cambridge, 1991.
- [16] G. D. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972.
- [17] A. W. Roscoe. Model-checking CSP. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*. Prentice-Hall, 1994.
- [18] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. of the 1995 IEEE Symp. on Security and Privacy*, pages 114–127, 1995.

# Verifying Cryptographic Protocols for Electronic Commerce

Dr. Randall W. Lichota

*Hughes Technical Services Company, P.O. Box 3310, Fullerton, CA 92834-3310*  
*lichota@stars1.hanscom.af.mil*

Dr. Grace L. Hammonds,

*AGCS, Inc., 91 Montvale Avenue, Stoneham, MA 02180-3616*  
*hammonds@dockmaster.ncsc.mil*

Dr. Stephen H. Brackin

*Arca Systems, Inc., 303 E. Yates St., Ithaca, NY 14850*  
*Brackin@arca.com*

## ABSTRACT

This paper describes the Convince toolset for detecting common errors in cryptographic protocols, protocols of the sort used in electronic commerce. We describe using Convince to analyze confidentiality, authentication, and key distribution in a recently developed protocol proposed for incorporation into a network bill-payment system, a public-key version of the Kerberos authentication protocol. Convince incorporates a "belief logic" formalism into a theorem-proving environment that automatically proves whether a protocol can meet its goals. Convince allows an analyst to model a protocol using a tool originally designed for Computer-Aided Software Engineering (CASE).

## 1.0 INTRODUCTION<sup>1</sup>

As electronic commerce on the Internet experiences explosive growth, so does the number of security protocols for safeguarding business transactions. Almost without exception, these protocols use cryptography, in the form of symmetric- and/or public-

key algorithms.<sup>2</sup> Using encryption does not guarantee protection, though. A protocol must be free of flaws that an electronic thief can exploit. Through such devices as clever replays and modifications of messages, legitimate parties to a protocol can be tricked into thinking they are communicating with each other when they are actually communicating with the thief.

While the use of formal methods does not necessarily result in detection of all such flaws, it increases the level of confidence in protocols for electronic commerce. This paper describes an automated toolset, Convince, that facilitates the analysis of cryptographic protocols by systematically checking a number of their essential security properties.

In general, cryptographic protocols use encryption to protect the confidentiality and/or integrity of message data, and to verify the identity of (i.e., *authenticate*) one or more of the parties involved in message transfers. To confirm that each message transfer in a protocol performs its intended security functions, one must ask questions such as the following:

---

<sup>1</sup> This work has been sponsored by the Air Force Materiel Command, Electronic Systems Center/Software Center (ESC/AXS), at Hanscom AFB, MA, and funded by Rome Laboratory, through contract numbers F19628-92-C-0006 and F19628-92-C-0008.

<sup>2</sup> Some of the more widely publicized protocols of this type include the Secure Sockets Layer (SSL), Secure Hypertext Transfer Protocol (S-HTTP), Private Communications Technology (PCT), and Secure Electronic Payment Protocol (SEPP). [BERN96]

- a. Can the sender be confident that the data being sent has the expected properties.
- b. Can the sender and receiver be confident that the confidentiality and integrity of the data are preserved in transit?
- c. Can the receiver be confident who sent the data?
- d. Can the sender later be confident that the intended party received the data sent?

Assuming that the cryptographic algorithms used are themselves relatively "safe"<sup>3</sup>, the answers to these questions depend on whether the parties to the protocol can *convince* themselves that the protocol provides the necessary assurances.

During the past decade, researchers have developed *belief logics* [BUR90, GON90, SYV94] that formalize inferences about what protocol parties "can be confident" of regarding authentication properties of protocols.<sup>4</sup> Constructing formal proofs from a belief logic thus gives a means of testing whether a protocol serves its intended functions.

Convince incorporates a belief logic into a specialized automatic theorem-proving environment. In this environment, a protocol designer or analyst uses Computer-Aided Software Engineering (CASE) tools as a front end to a formal theorem prover. Convince makes the formal verification process similar to debugging software. An analyst creates a protocol model (the "code"), specifies its associated initial conditions and goals (identifies the "code's" expected behavior), and makes incremental revisions to the model until the goals are either proved or the protocol is judged to be fatally flawed (the "code" executes correctly or is abandoned). Convince makes it possible to maximize the early detection of security-related design errors, without requiring a lot of theorem-proving expertise.

Convince's CASE-based interface is implemented using Interactive Development Environments' Software Through Pictures™ [IDE94a] system, which allows an analyst to model a protocol using a combination of familiar graphical and textual notations. Convince's proof process is implemented using a well-known Higher Order Logic (HOL) [GOR93] theorem prover.

We used Convince to analyze aspects of confidentiality, authentication, and key distribution in a recently proposed public-key version of the Kerberos authentication protocol, which the remainder of this paper will refer to as *PK Kerberos*. The PK Kerberos protocol is a component of the NetBill system for secure electronic commerce between on-line customers and merchants of on-line goods (e.g., reports) [COX95]. This protocol is being proposed as an Internet standard [CHU96].

Within NetBill, PK Kerberos is used to establish the initial authentication between customer and merchant. Consequently, we examined this protocol from two points of view: whether it is secure for the purpose for which it is intended (providing authentication services for NetBill); and whether it is reasonable for use in more general contexts (as would be expected for an Internet standard).

This work is part of a series of efforts, begun under the Air Force's Portable, Reusable, Integrated Software Modules (PRISM) program, to identify emerging technologies that are ready to be incorporated into ongoing Air Force programs. Convince development came after the review of a Rome Laboratory research prototype, the Romulus Verification Environment [ORA94]. This review clearly established the value of protocol analysis based on belief logic, but in order to effectively interact with Romulus, the user had to have specialized knowledge of its HOL-based theorem-proving environment. We quickly recognized that the effort needed to acquire this specialized knowledge would limit user acceptance. We also considered other protocol analysis tools, described in Section 5, but each of these also had serious limitations.

The remainder of this paper is organized as follows: Section 2 gives an overview of Convince's theoretical foundation, its belief logic; Section 3 gives an overview of Convince's software components; Section 4 describes using Convince to model and analyze PK Kerberos; Section 5 gives an overview of related work; and Section 6 gives our conclusions and recommendations for future work.

## 2.0 CONVINCE'S BELIEF LOGIC

Like all other belief logics, the Convince belief logic grew out of the BAN logic developed by Burrows, Abadi, and Needham [BUR90]. In the BAN logic, an

---

<sup>3</sup> The strength of encryption algorithms is not covered by Convince.

<sup>4</sup> While the emphasis in belief logics is on authentication, their rules implicitly address basic aspects of confidentiality and integrity.

authentication protocol is transformed into a sequence of logical statements that are then analyzed.

Gong, Needham, and Yahalom developed another belief logic, the GNY logic, based on BAN but expressed at a lower level of abstraction [GON90]. This makes it able to identify a somewhat larger class of protocol flaws.

Gong then discovered that it is possible to specify and "verify" protocols, using the original GNY logic, that are impossible or unreasonable to implement, resulting in situations where the causality of beliefs is not preserved [GON91]. He developed conditions for excluding these "infeasible" protocols.

The Romulus prototype [ORA94] implemented part of the GNY logic, in HOL, and implemented Gong's refinement to the original GNY logic.

Brackin [BRA96a] subsequently developed a HOL implementation of the full GNY logic, including Gong's refinement, and developed logics extending this logic. One of these extensions, called BGNY, is the foundation for the Convince toolset. It covers protocols using symmetric- and public-key encryption, ordinary and key-dependent hash codes, key-exchange algorithms, multiple encryption and hash algorithms, and protocols using hash codes as keys.

At a high level, BGNY is a set of rules identifying the conditions under which protocol participants can obtain data and draw conclusions about this data and other protocol participants. While most of the BGNY rules are based on GNY, there are omissions, additions, and modifications. The omitted rules reflect making more restrictive use of the concepts of "conveyance" and "trust" (see Table 1 below). The new and modified rules implement extensions to the GNY logic, remove unnecessary restrictions in the GNY logic, and correct errors in the GNY logic [BRA96a].

In the following informal descriptions of sample BGNY rules, the rules describe how a principal B can obtain data sent in encrypted form:

#### Rule P1:

If B receives a message M, then B possesses M.

#### Rule P4:

If B possesses a decryption algorithm and a key, then B possesses the result of applying this decryption algorithm, with this key, to any message it possesses.

#### Rule P7:

If B possesses the result of applying a decryption function, with a key, to a message encrypted with the corresponding encryption function and key, then B possesses the decrypted message.

While a complete description of the BGNY logic is beyond the scope of this paper, Table I lists the logical statements and symbols used in the discussions that follow. These constructs are part of Convince's Intermediate Specification Language (ISL) [BRA97]. ISL is used to describe protocols and their expected authentication properties, as well as their principals and these principals' initial conditions.

## 3.0 CONVINCE SOFTWARE

Three major software tools lie at the heart of Convince: the Software through Pictures™ (StP), version 2.0, Object Modeling Tool (OMT), a Higher Order Logic (HOL) theorem prover, and a translator, based on LEX and YACC, to convert ISL specifications into HOL specifications.<sup>5</sup> Figure 1 depicts the process, and the data flow between software components, when a user analyzes a protocol. The dashed lines show where user input is required. As the figure indicates, once the protocol is specified, most of the remaining work is done automatically.

From a textual or other description of the protocol, the user creates a model — a high-level representation — under StP/OMT. This model identifies the important attributes of the principals, messages, and encryption services (e.g., keys and other parameters), used in the protocol.

From this model, Convince generates an ISL specification, which provides a representation of all the defined elements of the protocol. Convince translates the ISL representation into an internal HOL specification, processes the HOL specification to create a HOL theory of the protocol, and executes a set of functions that automatically make deductions in this theory from the rules in the BGNY logic.

Convince produces screen output telling whether it proved all the goals. If it cannot automatically prove a goal, Convince displays the goal to the user and terminates its theorem-proving process. In this case, it

<sup>5</sup> The Convince components are hosted on Sun SPARCstation platforms, running SunOS 4.1.3.

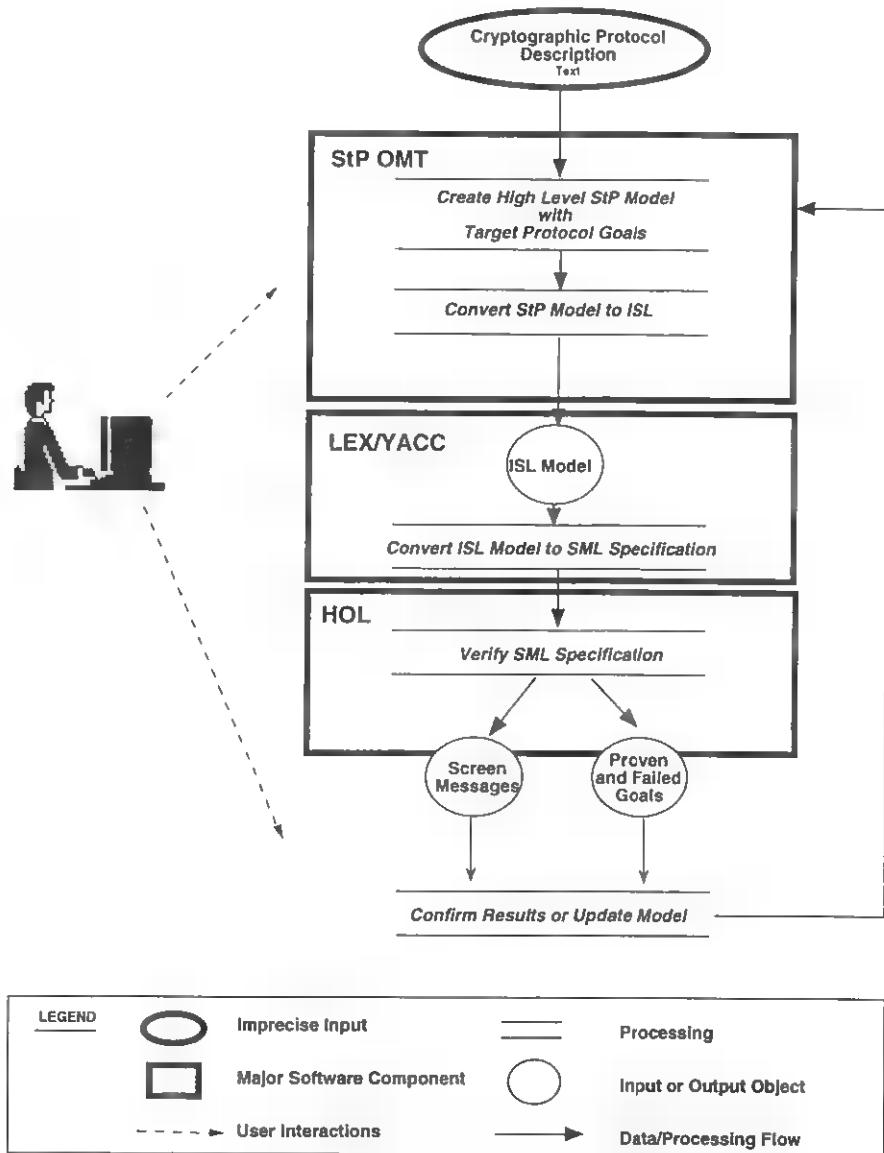


Figure 1. Convince Process and Data Flow

produces ISL output files describing proved and unproved goals and subgoals.

A goal failure can indicate that either the protocol's specified initial conditions are insufficient, or there is some other error in the formal description of the protocol, or the protocol is flawed. The user can repeat the model/analyze process, making changes to the model, until becoming satisfied that either the protocol is flawed or does not have a flaw detectable by the BGNY logic.

### 3.1 StP/OMT COMPONENT

StP/OMT is a CASE tool that facilitates the development of software systems via the Object Modeling Technique (OMT) methodology [IDE94b]. The latter was originally developed by James Rumbaugh and his associates at General Electric in the late 1980's. More recently, this methodology has been enhanced through the efforts of Jacobson, Booch, and Rumbaugh. While the methodology has focused primarily on the development of custom software implementations, we adapted it for the modeling and analysis of authentication protocols. In the process, we enhanced the methodology by employing a somewhat richer notation necessary for describing the salient characteristics of authentication protocols.

StP/OMT automates the application of the OMT methodology by providing a set of graphical editors that can be used to create a model of an authentication protocol, providing views of the model at different levels of abstraction. We describe three below: Use Cases, Event Traces, and Dynamic Models.

Use Case models are an extension to the original OMT formalism. We adapted this notation to provide a structure for organizing protocol specifications. We use Event Trace diagrams to describe the sequence of message transfers that comprise an authentication protocol. Each Event Trace consists of a context object, a set of object classes, and a set of directed line segments that denote message transfers. The user labels each message transfer with text denoting the nature of the message (e.g., "request for public key") and the protocol stage at which the transfer occurs. Dynamic models are used to depict the state of each principal between message transfers.

In order to completely describe the properties of authentication protocols, we had to extend the notation provided by OMT. We did this primarily by using annotations. An annotation represents additional protocol information that is associated with StP/OMT model elements. The model elements that require annotations include principals, message transfers, context objects, and states.

Annotations associated with context objects represent definitions of cryptographic and hash functions, keys, principal names, and other variables (e.g., timestamps and nonces).

Annotations associated with a principal's state correspond to ISL statements. In the case of start states, annotations represent initial conditions assumed to be

true at the start of a protocol execution. Because they represent initial conditions, these annotations are limited to statements constructed from the Received and Believes operators. Annotations associated with other states (i.e., intermediate and end states) are not so restricted; these may be composed from any of the ISL statements.

Examples of an Event Trace diagram, Dynamic Model, and associated annotations are given in Figures 2 and 3.

### 3.2 LEX/YACC

LEX and YACC are standard UNIX utilities used to implement a parser to convert formats inside Convince. ISL is a textual language whose syntax is a superset of the annotation syntax employed under StP/OMT. ISL specifications are generated from a Convince model via a simple command option invoking the parser. ISL specifications have four major sections:

1. A set of definitions for certain data types, including principals, algorithms, and keys;
2. A set of initial conditions, indicating data items and beliefs of principals;
3. A sequence of message transfers denoting the protocol steps, or stages; and
4. A set of goal conditions showing what the protocol should achieve from the point of view of the principals.

An annotation associated with a principal denotes the name to use for the principal in message descriptions, initial conditions, and goal statements. This allows one to use longer, more descriptive principal names in OMT diagrams, while using shorter, equivalent names in formulas.

For message transfers, annotations represent the structure of messages conveyed between principals.

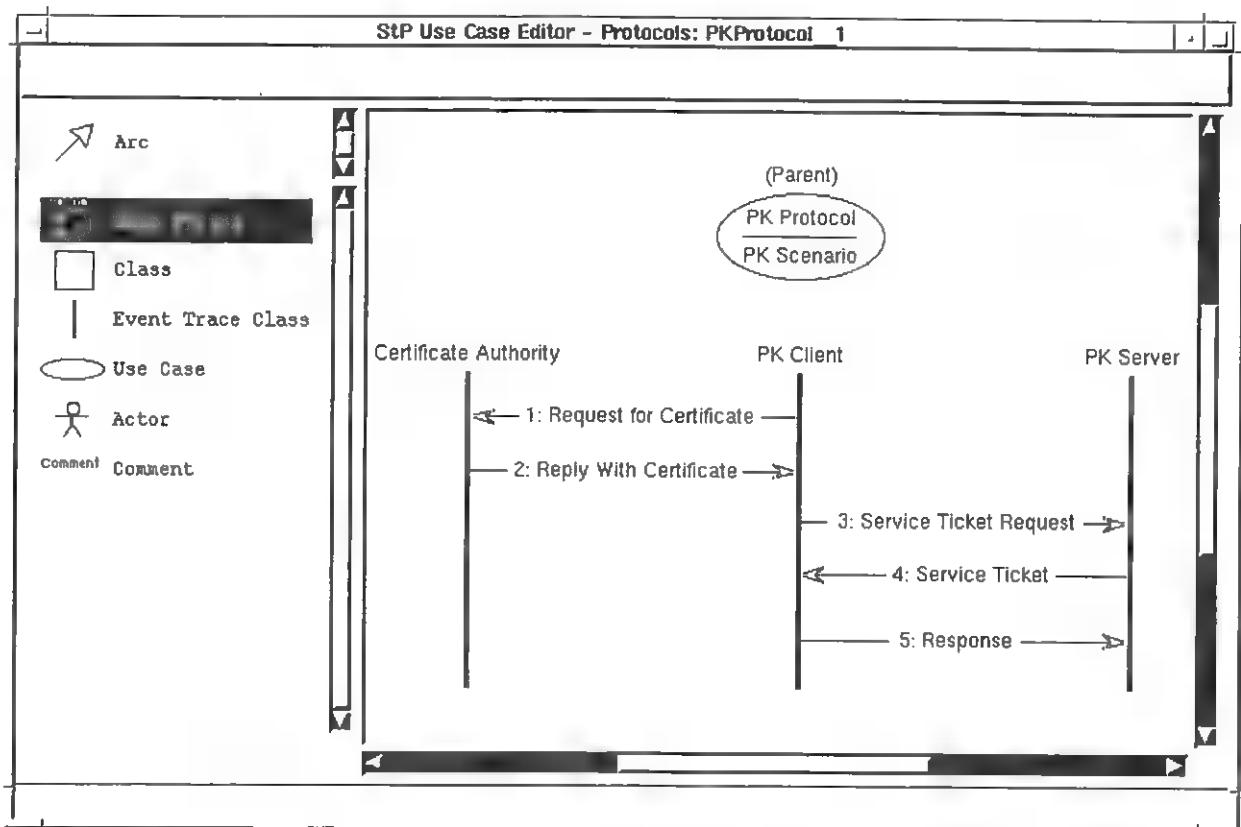


Figure 2. Event Trace Diagram

Goal conditions are numbered according to the transfer stages defined in the protocol model. The number of a goal condition is the stage of the message transfer expected to cause the goal to become true.

An example of a complete ISL specification for PK Kerberos is given in Appendix A.

In Convince, verification of an authentication protocol uses Higher Order Logic (HOL). This necessitates translating the ISL specification into a HOL internal form prior to the actual proof process. The LEX/YACC translator makes this translation. It produces HOL code that defines a theory of the protocol and invokes the automatic proof process.

### 3.3 HOL COMPONENT

The core of Convince is the Highter Order Logic (HOL) implementation of the BGNY logic together with a proof procedure that automates the construction of proofs in this logic. The proof procedure checks whether the protocol's goals follow from the protocol's

definition and the rules of the BGNY logic. If a goal's proof fails, the problem might be an error in the initial assumptions, an overly ambitious goal, or a security flaw in the protocol.

Convince's output files listing proved and unproved goals and subgoals, in ISL, help identify the cause of proof failure.

## 4.0 EXAMPLE: PK KERBEROS

To illustrate how Convince can be used to model and analyze cryptographic protocols that support electronic commerce, we provide the example of PK Kerberos [CHU96], a public-key version of the Kerberos authentication protocol [STE88].

All versions of Kerberos seek to establish secure communication between two parties while maintaining confidentiality and data integrity and detecting masquerading and replays. In earlier versions of Kerberos, a centralized Key Distribution Center (KDC) authenticates a user through symmetric-key encryption, then gives this user a shared key for subsequent

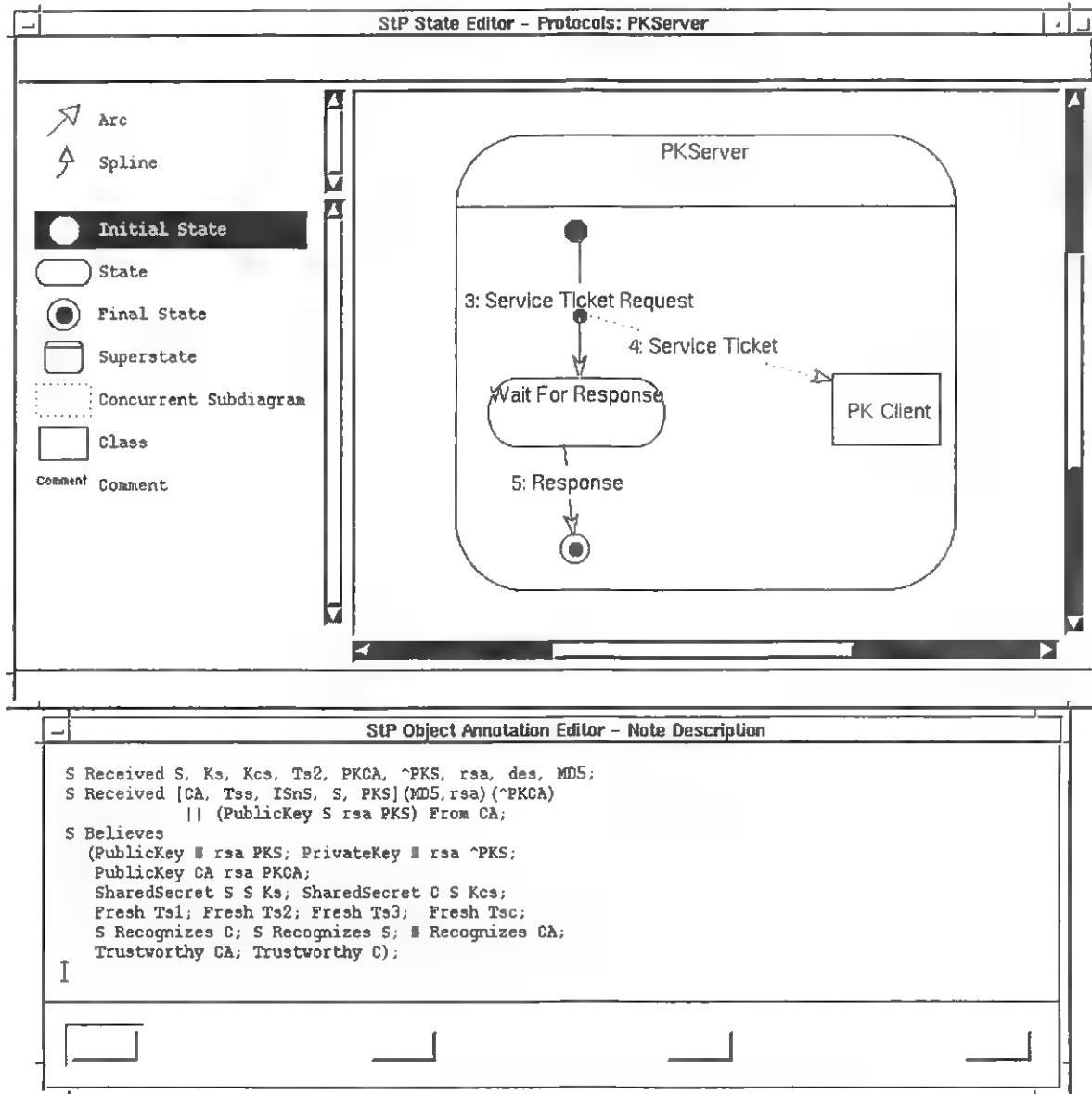


Figure 3. Dynamic Model and Annotations

communications with other parties. This makes the KDC a potential bottleneck in the system, as well as a single point of failure that could disrupt the entire system if compromised.

PK Kerberos attempts to overcome this weakness by employing Public Key Certificates based on the X.509 standard [CCI88].<sup>6</sup> After the initial authentication, PK

Kerberos continues as Kerberos does, with the exchange of symmetric keys to be used for later communication.

#### 4.1 PK KERBEROS PROTOCOL

<sup>6</sup> Full implementation of these certificates will later involve an infrastructure to support the creation and initial

distribution of these certificates, but they are available today for both public and private users.

The PK Kerberos protocol involves three parties: a client C, a server S, and a certificate authority CA.<sup>7</sup> Initially, C requests S's public-key certificate from CA. In a series of message exchanges, C receives S's public key from CA, then, using this public key along with its own private key, requests and obtains a symmetric key for later use. By the end of the exchange, both C and S can believe that they have correctly identified each other, using certificates that they trust, and the key they share is known only to themselves.

In the model of PK Kerberos shown below, we have excluded certain fields that would normally be present in the protocol and in X.509 certificates: message IDs; encryption, signature, and message-digest algorithms; version numbers; compromise key lists; and certificate serial numbers. While these fields are needed for an implementation, they are not relevant for determining the security properties of interest, i.e., confidentiality, integrity, and authentication. In another simplification, we leave out the validity periods for keys, assuming that the protocol is running when the keys are valid.

The protocol's description uses the following terms, along with the BGNY/ISL notation in Table 1:

C	Client
S	Server
CA	Certification Authority
CertificateX	Public-key certificate of X, defined below, signed by an authorized Certification Authority
Ts#	Time stamp number #; Ts1 is also a proxy for current validity interval
Kr	Symmetric key to be used as a one-time session key
Kcs	Symmetric key to be used as a long-term session key
Ks	Symmetric key known only to S and used to protect tickets
PKC, PKS	Public keys for C and S
$\wedge$ PKC, $\wedge$ PKS	Private keys for C and S
MD5	Hashing algorithm
rsa, des	Public- and symmetric-key encryption /decryption algorithms

<sup>7</sup> The inclusion of the CA is optional; the source for S's public-key certificate could be S itself. For the purpose of this analysis, we use CA as both the repository for certificates and the authority that verifies their integrity. This option allows us to explore issues of levels of trust, with CA having the highest level.

"authdata" is defined as data used to help authenticate C to S:

$$\text{authdata} = S, \text{Certificate}_S, T_{s1}, Kr$$

The public-key certificate for a principal X is defined as follows:

$$\text{Certificate}_X = CA, T_{s\#}, X, PK_X$$

$$\{H(CA, T_{s\#}, X, PK_X)\}rsa(^{PKCA})$$

"CA" is the certification authority for the certificate; CA serves as the certificate repository in our model.

The transactions in PK Kerberos are as follows:

1. C requests S's public-key certificate; C could request it directly from S, but in our model asks CA.
2. C receives the requested public-key certificate.
3. C uses S's public key to encrypt a new temporary symmetric key, Kr, for one-time use by S, along with C's own public-key certificate and a signature created by encrypting the hash of Kr along with S's public-key certificate and a timestamp. The ISL statement associated with this signature asserts that C believes Kr will be known only to itself and S.
4. S decrypts the message to obtain Kr, and checks the signature to confirm that Kr came from the C named in the enclosed certificate. S creates a long-term symmetric key, Kcs, for itself and C, and sends it, encrypted under Kr, back to C. S also sends a "ticket" with Kcs, C's name, a timestamp, and possibly other security information not shown in the model (e.g., file access rights). S encrypts this "ticket" with Ks; C is to return this encrypted ticket when making later requests from S.
5. C returns a timestamp encrypted with Kcs to confirm that it received Kcs. C also returns the encrypted ticket for additional validation.

## 4.2 INITIAL CONDITIONS AND GOALS

The initial conditions for this protocol consist of all the received items and beliefs that the analyst assumes are held by the principals at the start of the protocol. Typical initial conditions are that the principals hold their own public and private keys, and that they trust the appropriate authority that dispenses these keys. A complete list is included in Appendix A.

Goal conditions should express the underlying purpose of the protocol's exchanges, such as that the principals believe they each possess a common symmetric key. The following shows the major goals for PK Kerberos. The numbers represent the protocol stages after which the associated goals should be true.

2. C Believes PublicKey S rsa PKS;
3. S Possesses Kr;  
S Believes  
(SharedSecret C S Kr;  
C Possesses Kr;  
C Believes SharedSecret C S Kr);
4. C Possesses Kcs;  
C Believes  
(SharedSecret C S Kcs;  
S Possesses Kcs;  
S Believes SharedSecret C S Kcs);
5. S Believes  
(C Possesses Kcs;  
C Believes SharedSecret C S Kcs;  
SharedSecret C S Kr;  
C Possesses Kr;  
C Believes SharedSecret C S Kr);

After the second transaction, for instance, C should have reason to believe that it has a bona-fide public key for S. By the third transaction, S should possess the session key (Kr) that it believes is a shared secret between itself and C. By the last step, 5, S should believe C holds the shared symmetric key Kcs.

### 4.3 CONVERTING DESCRIPTIONS TO StP

From a description of the protocol, usually text, the user creates a protocol model by first defining the protocol elements within StP/OMT. The user then constructs a Use Case diagram and associates it with a specific protocol scenario. In our example, we model only a single scenario, shown in the Event Trace diagram in Figure 2. StP's Event Trace editor automatically provides a context object, here labeled as PK Protocol. The user next adds the vertical bars representing the principals, and labels them accordingly. The user adds a set of directed line segments to denote the message transfers that occur as part of the protocol scenario, and labels each message transfer with a text string denoting the nature of the message (e.g., "request for public key") and the stage of the protocol at which the transfer occurs.

After completing the Event Trace Diagram, the user constructs a Dynamic Model for each of the principals. As shown in Figure 3, the Dynamic Model for S in PK Kerberos is a state transition sequence. The start state is represented as a solid circle, the intermediate state as a rounded rectangle, and the end state as a bull's eye. Transitions between states are represented by directed

lines whose labels denote the received events responsible for triggering the transitions. Message transfers that are produced by the principal are represented as output events. These are associated with directed lines connecting a state transition to the principal who is the recipient of the message.

Generally speaking, the start state of a Dynamic Model corresponds to a subset of the initial conditions for the protocol. Accordingly, for each start state, the user provides annotations that represent the initial conditions of the corresponding principal. In Convince, these conditions are limited to statements of belief or reception. The initial conditions of principal S are shown as a sequence of ISL statements at the bottom of Figure 3.

After adding the initial conditions to the model, the user provides annotations for the intermediate and end states. These annotations represent goals for the protocol (e.g., C and S share a certain symmetric key), which should become true once the protocol reaches a specific state.

### 4.4 CONVERTING StP TO ISL

Once the initial conditions, transactions, and goals have been input, the user directs Convince to convert the model to an ISL specification, then invoke the translation and proof processes. This is done with a single menu selection from StP/OMT. The full ISL specification for PK Kerberos is given in Appendix A.

The LEX/YACC and HOL subsystems of Convince can be used without Convince's StP interface. To do so, the user prepares an ISL specification directly, as a text file, and gives the name of this file as a command-line argument to the LEX/YACC translator, which invokes the proof process.

### 4.5 RUNNING THE VERIFIER

Convince attempts to verify a model by proving that it meets both its user-specified goals and a standard set of goals, originally derived from the GNY logic, that encompass all protocol properties that are typically of interest [BRA96b].

During the first few iterations of creating or modifying a protocol model and seeing if Convince

proves that it meets its goals, proof failures will typically result from insufficient initial conditions, such

as a principal not possessing a needed algorithm. This was the case with our analysis of PK Kerberos. Insufficient initial conditions relating to possession often result in protocol feasibility failures (i.e., a principal attempting to send something it does not possess) [GON91].<sup>8</sup>

In PK Kerberos, the most significant proof failure due to an insufficient initial condition involved S's having to trust C to create the temporary symmetric key Kr. Our original model did not include this condition, and the proof failed at the subgoal of S believing Kr is a shared secret. Even though this key is for one-time use, production of weak or guessable keys by C could cause vulnerabilities in the protocol. Within the context of NetBill, C will be executing software with a predefined algorithm for creating these temporary keys, which is expected to limit their vulnerability. In more general contexts, this assumption should be examined closely. Problems due to insufficient initial conditions are generally easy to correct once the reason for proof failure is identified. Convince's output files giving lists of proved and unproved standard goals, and their proved and unproved subgoals, are useful for this purpose. It should be noted, however, that some initial conditions might impose constraints on an implementation that are unacceptable.

In addition to problems that result from insufficient initial conditions, proof failures can result from inadequate or inappropriate associations of properties, expressed via ISL statements, with messages. As a rule of thumb, encrypted messages used to convey keys that are shared secrets should include an associated statement expressing this fact.

We call the types of errors noted above *setup* errors because they are due to the specific form of the model being constructed and do not necessarily show flaws in the protocol itself. Similarly, apparently redundant information in a protocol, which we found in the PK Kerberos example, might not cause security flaws.

In translating the English descriptions of the PK Kerberos example into ISL, we uncovered a particular aspect of the protocol that demonstrated the need for one of our extensions to the GNY logic. In stage 4, S

<sup>8</sup> Additional forms of insufficient initial conditions we encountered in modeling other protocols include beliefs relating to "freshness" (e.g., recent timestamps), recognition of key message elements (e.g., principal names), trust, and properties of keys (e.g., that a principal's public key is believed to be what it is).

sends out an encrypted copy of a ticket that only S can decrypt, along with the same and more information in a form that is readable by C. In stage 5, C uses the information available to it to prepare an appropriate authenticator, and sends that authenticator, along with the ticket that only S can decrypt, back to S. This is necessary because S has forgotten everything except the key it used to encrypt "send this back to me" copies of the tickets it has sent out in the last few hours. S uses this key to decrypt these tickets when they are sent back to it, to confirm that they were originally from S and go with the authenticators sent back with them. So rather than remember each ticket or a hash of this ticket, S remembers only the key Ks it uses to encrypt these tickets.

This is not expressible in the GNY logic, which assumes that principals remember everything for the length of a protocol run; every principal has perfect memory of the messages it has sent or received. For a potential attacker, this is a good, conservative assumption, but for legitimate protocol principals, PK Kerberos shows that it might not be true.

In total, it took us about 3 days of tool use, spread over a couple of weeks, to resolve all the problems in our model of PK Kerberos. Once the model was finished, the conversion to ISL and production of all the proofs took less than 5 minutes on a Sun SPARCstation 20. In the course of our analysis, we proved that by the end of PK Kerberos the keys are securely in place with the parties authenticated to each other, but this requires that the client be trusted to create a sufficiently strong symmetric session key. We concluded that the protocol contains elements that, while appropriate for NetBill, might be unnecessary or insufficient for use in other contexts. For example, in some environments, encryption keys should only be generated by a high-integrity source.

## 5.0 RELATED WORK

Romulus [ORA94] represents an early effort to automate the analysis of authentication protocols via theorem proving. Romulus implements belief logic, in HOL, in the form of a theory of authentication, crypto\_90. This implementation requires that a user create protocol models in HOL, with all initial assumptions, protocol actions, initial conditions, and goals expressed as HOL statements. The user produces proofs by applying HOL tactics, by hand, using rules defined in crypto\_90. A typical verification strategy is first proving a set of simple conditions that can be

applied repeatedly as lemmas in the proof of the goals. Romulus depends heavily on the HOL environment, which necessitates that a user be thoroughly familiar with HOL. In addition, crypto\_90 is limited to symmetric-key encryption; it does not handle public-key encryption, signatures, hash functions, key-exchange functions, or the use of hash codes as keys.

More recently, Tom Schubert and his students at Portland State University [SCH95] created a HOL implementation of a variant of the SVO belief logic [SYV94]. The SVO logic represents an effort to harmonize different belief logics into a single unified formalism. Schubert and his students used the resulting tool to specify and verify the correctness of key escrow protocols. As a byproduct of this work, they created an infrastructure to support interactive proofs in HOL. They have made plans to extend this tool by incorporating semi-automated decision procedures.

The NRL Protocol Analyzer [MEA91] and the MITRE Interrogator [MIL84] take somewhat different approaches to protocol analysis. They are rewrite systems that determine whether an undesirable state can be reached from a given set of initial states. An sample use of these tools would be determining whether any amount of manipulation of exposed data by an intruder could obtain a key that should be kept secret.

The NRL Protocol Analyzer differs from Convince in the type of goals it can handle; it attempts to show the absence of all possible flaws in a protocol, while Convince attempts to establish that certain beliefs are justified. The NRL Protocol Analyzer is a low-level tool that requires a user to define the undesirable states.

The Interrogator, which was built by Millen, generates a large number of paths through the protocol ending in an insecure state, and sees whether any of these begin in a valid initial state. It does not prove that there are no such paths, and finds flaws similar to those found by the NRL Protocol Analyzer.

The Interrogator and NRL Analyzer are in principal capable of finding more flaws in cryptographic protocols than Convince is, but they are drastically much slower and drastically more difficult to use. Protocol analyses using them typically take weeks or months of effort by these tools' developers.

The tools described above are tailored specifically for analyzing cryptographic protocols. While in theory one could apply belief logics to an analysis of the strengths and weaknesses of non-cryptographic protocols, this

would require changing the rules that implicitly rely on properties of cryptographic functions.

Still, to varying extents, these tools share their underlying technologies with verification tools implemented for other domains; see [BUT95] for a description of some recent developments. While these other tools are too numerous to describe here, two are noteworthy. One of these, Murφ, incorporates a protocol description language and verifier for detecting design errors such as deadlocks [DIL92]. Another recent tool combines the formal language Estelle with Numerical Petri Nets (NPN) [JIR95]. In this system, protocols are first specified in Estelle and then translated into NPN specifications. These can then be analyzed using an NPN verifier.

## 6.0 SUMMARY & FUTURE DIRECTIONS

This paper has described a tool for modeling and analyzing authentication protocols. As the tool has matured, we have used it to analyze aspects of confidentiality, authentication, and key distribution in a recently proposed protocol, a public-key version of Kerberos. Our analysis brought questions regarding initial-conditions assumptions to light, including an issue of trust. Whether this is a problem depends on how the protocol is actually implemented and used.

We have provided this information to the authors of the PK Kerberos protocol and will support them with subsequent analyses if they change the protocol.

In general, the viability of proof-based analyses depends on the reasonableness of the proof goals, the scope of the logic, and whether a sufficient set of initial conditions are available. Convince has evolved to address the salient issues relating to the authentication protocols we have analyzed.

In translating English descriptions into ISL, we have been able to quickly identify those aspects of protocols that are not clear. We are following ongoing efforts in the protocol analysis community to develop a standard protocol specification language.

In the future, we plan to extend Convince to provide more detailed guidance, through its graphical user interface, for identifying the cause of proof failures. We also plan to explore how Convince might be used in conjunction with other types of protocol analysis tools.

## 7.0 ACKNOWLEDGMENTS

The authors would like to thank John Chuang and Marvin Sirbu, of the Carnegie Mellon University Engineering and Public Policy Department, for allowing us to use a preliminary version of the Public Key Kerberos protocol. We would also like to credit Rae Burns and Yi-Fang Koh, whose use of StP as a front end to a design tool for multilevel secure databases was the inspiration for our use of StP. We also express our appreciation to Jack Wool and Joe Giordano for their support of this effort.

## 8.0 REFERENCES

[BERN96] T. Bernstein, A. B. Bhimani, E. Schultz, and Carol A. Siegel, *Internet Security for Business*. New York: Wiley Computer Publishing, 1996, pp. 251-255.

[BRA96a] S. H. Brackin, "A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols". *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, County Kerry, Ireland, 1996.

[BRA96b] S. H. Brackin, "Deciding Cryptographic Protocol Adequacy with HOL: The Implementation". *International Conference on Theorem Proving in Higher Order Logics*, 1996.

[BRA97] S. H. Brackin, "An Interface Specification Language for Automatically Analyzing Cryptographic Protocols". To be presented at the Internet Society Symposium on Network and Distributed System Security, February 1997.

[BUR90] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication". *ACM Transactions on Computer Systems*, 8(1), 1990.

[BUT95] R. Butler, J. Caldwell, V. Carreno, C. Holloway, P. Miner, and B. Di Vito, "NASA Langley's Research and Technology Transfer Program in Formal Methods". In *Proceedings of the Tenth Annual Conference on Computer Assurance*, pages 135 - 150, 1995.

[CCI88] CCITT, Recommendation X.509: The Directory Authentication Framework, 1988.

[CHU96] J. Chuang and M. Sirbu, Internet Draft, Update to RFC 1510, "Public-Key Based Ticket Granting Service in Kerberos", Carnegie Mellon University/INI, draft-sirbu-kerb-ext-00.txt, May 6, 1996.

[COX95] Benjamin Cox, J. D. Tygar, Marvin Sirbu, "NetBill Security and Transaction Protocol", *Proceedings of the First Workshop on Electronic Commerce*, 1995.

[DIL92] D. Dill, A. Drexler, A. Hu, and C. Yang, "Protocol Verification as a Hardware Design Aid", 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1992.

[GON90] L. Gong, R. Needham, and R. Yahalom, "Reasoning about Belief in Cryptographic Protocols". In the *Proceedings of the 11th IEEE Symposium on Research in Security and Privacy*, pp. 234-248, 1990.

[GON91] L. Gong, "Handling infeasible specifications of cryptographic protocols". *Proceedings of the Computer Security Foundations Workshop IV*, pp. 99 - 102, 1991.

[GOR93] M. Gordon and T. Melham, *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*, Cambridge, UK: Cambridge University Press, 1993.

[IDE94a] Interactive Development Environments, *Fundamentals of StP, Release 1*, February 1994.

[IDE94b] Interactive Development Environments, *Creating OMT Models, Release 1*, February 1994.

[JIR95] A. Jirachaisattana and R. Lai, "An Estelle-NPN Based System for Protocol Verification", *Proceedings, 10th Annual Conference on Computer Assurance*, pp. 135 - 150, 1995.

[MEA91] C. Meadows, "A System for the Specification and Analysis of Key Management Protocols". In *Proceedings of the Symposium on Security and Privacy*, 1991, pp. 182-147.

[MIL84] J. Millen, "The Interrogator: A Tool for Cryptographic Protocol Analysis". In *Proceedings of the Symposium on Security and Privacy*, pp. 134-141, 1984.

[ORA94] Odyssey Research Associates, *Romulus User's Manual*, March 1994.

[SCH95] T. Schubert and S. Mocas, A Mechanized Logic for Secure Key Escrow Protocol Verification, *International Workshop on Higher Order Logic and its Applications*, 1995.

[STE88] J.G. Steiner, C. Newman, and J. I. Schiller, "Kerberos: An Authentication Service for Open Network Systems". In Proceedings of the USENIX Winter Conference, 1988, pp. 191 - 202.

[SYV94] P. Syverson and P. van Oorschot, "On Unifying Some Cryptographic Protocol Logics". In *Proceedings of the Symposium on Security and Privacy*, Oakland, CA, 1994, pages 14-28.

## APPENDIX A. PK KERBEROS ISL SPECIFICATION

### DEFINITIONS:

PRINCIPALS: C, S, CA;  
 SYMMETRIC KEYS: Kr, Kcs, Ks;  
 PUBLIC KEYS: PKC, PKS, PKCA;  
 PRIVATE KEYS: ^PKC, ^PKS, ^PKCA;  
 OTHER: Ts1, Ts2, Ts3, Tsc, Tss;  
 ENCRYPT FUNCTIONS: des, rsa;  
 HASH FUNCTIONS: MD5;  
 des WITH ANYKEY HASINVERSE des WITH ANYKEY;  
 rsa WITH ^PKCA HASINVERSE rsa WITH PKCA;  
 rsa WITH ^PKC HASINVERSE rsa WITH PKC;  
 rsa WITH PKS HASINVERSE rsa WITH ^PKS;

### INITIAL CONDITIONS:

CA Received rsa, MD5, CA, S, Tss, PKS, ^PKCA;  
 CA Believes PublicKey S rsa PKS;  
 C Received  
 rsa, des, MD5, C, S, Ts1, Ts3, Kr, PKCA, ^PKC;  
 C Received  
 [CA, Tsc, C, PKC] (MD5, rsa) (^PKCA) || (PublicKey S rsa PKC) From CA;  
 C Believes  
 (PublicKey CA rsa PKCA;  
 SharedSecret C S Kr;  
 Fresh Ts2; Fresh Tss;  
 C Recognizes C; C Recognizes S;  
 Trustworthy CA; Trustworthy S);  
 S Received  
 des, rsa, MD5, Ts2, Ks, Kcs, PKCA, ^PKS;  
 S Received  
 [CA, Tss, S, PKS] (MD5, rsa) (^PKCA) || (PublicKey S rsa PKS) From CA;  
 S Believes  
 (PrivateKey S rsa ^PKS; PublicKey CA rsa PKCA;  
 SharedSecret S S Ks; SharedSecret C S Kcs;  
 Fresh Ts1; Fresh Ts3; Fresh Tsc;  
 S Recognizes C; S Recognizes S;  
 Trustworthy CA; Trustworthy C);

### PROTOCOL:

1. C -> CA: S;
2. CA -> C:

[CA, Tss, S, PKS] (MD5, rsa) (^PKCA) || (PublicKey S rsa PKS);  
 3. C -> S:  
 S, Ts1,  
 {Kr, C,  
 [CA, Tsc, C, PKC] (MD5, rsa) (^PKCA) || (PublicKey C rsa PKC),  
 [S, Ts1, Kr,  
 [CA, Tss, S, PKS] (MD5, rsa) (^PKCA) || (PublicKey S rsa PKS)  
 ] (MD5, rsa) (^PKC) || (SharedSecret C S Kr)  
 }rsa(PKS);  
 4. S -> C:  
 S,  
 {Kcs, C, Ts2}des(Ks),

(C, S, Kr, Kcs, Ts2)des(Kr) || (SharedSecret C S Kcs);  
 5. C -> S:  
 S,  
 {Kcs, C, Ts2}des(Ks),  
 (C, Ts3)des(Kcs) || (SharedSecret C S Kcs);

### GOALS:

2. C Believes PublicKey S rsa PKS;
3. S Possesses Kr;  
 S Believes  
 (SharedSecret C S Kr;  
 C Possesses Kr;  
 C Believes SharedSecret C S Kr);
4. C Possesses Kcs;  
 C Believes  
 (SharedSecret C S Kcs;  
 S Possesses Kcs;  
 S Believes SharedSecret C S Kcs);
5. S Believes  
 (C Possesses Kcs;  
 C Believes SharedSecret C S Kcs;  
 SharedSecret C S Kr;  
 C Possesses Kr;  
 C Believes SharedSecret C S Kr);



# Legal Signatures and Proof in Electronic Commerce

Benjamin Wright, Attorney and Author: *The Law of Electronic Commerce*

## Abstract

*Legally acceptable methods for signing electronic documents are still under development. To enable dual-key digital signatures, Utah has enacted a scheme for certifying a public key through a licensed certification authority. The Utah scheme concentrates risk in the private key. In contrast, the IRS is using a biometric signature technology called PenOp, which tries to spread risk such that no particular feature of the signing event carries much weight.*

## I

### EGGS IN BASKETS: DISTRIBUTING THE RISKS OF ELECTRONIC SIGNATURES

Summary: The risks with electronic signatures might be addressed with any number of alternative strategies. One strategy, designed to appeal to the common person, would use biometrics to spread the risks so that no particular feature of the signing process, such as a private key, is highly important.

Electronic commerce brings questions about how to sign, or evidence legal approval of, electronic documents. Evidence that a person approved a particular electronic document might be gathered many different ways. The article evaluates two ways, one using public-key cryptography and the other using pen biometrics.<1>

The signing of a document is a social event, not a scientific event. It is an act in which an individual (Alex) evinces approval of the document so that someone else (Bob) can perceive that approval, understand it, and later prove it to other people. But the bonding of Alex to the document is never a perfectly reliable process. Whatever evidence exists to support the bond is subject to challenge. In other words, signing documents involves risk.

### TRADITIONAL INK AND PAPER STRATEGY

Many risks afflict the traditional signing of a paper document. First, there is no standard method for signing in ink. Alex is not taught or required to sign documents in a forensically reliable way. Alex is free to sign in any way he chooses, and to change his signature from minute to minute. For any given signing, Alex is free if

he so desires to use as his signature any strange and indecipherable scribble.

Whether any given document signed by Alex does or does not contain Alex's usual, verifiable signature is for all practical purposes a secret. Rarely is Alex's signature compared against specimens to confirm authenticity.

Ink signatures can be forged. There is no guarantee that any given ink signature can be verified by forensic science. Science can only offer an educated opinion as to whether the signature is authentic, and it can do so only under the right circumstances (including, for example, the availability of several good specimen signatures).

Other risks impede the linking of Alex to a given paper document. If the document is multiple pages in length, one or two of the pages could be switched after the document was signed. There is even the risk that the document is organized in an ambiguous or confusing way, so that an observer cannot discern for certain which parts of the document Alex agreed to and which parts he did not.

These risks mean that in the event of a dispute,<2> it is not always easy to tie Alex to specific words in a paper document. When Alex signs a document and gives it to Bob, Bob is not guaranteed that he will later be able to prove that Alex signed it.

Alex might raise any number of objections to repudiate the document. Conversely, Alex is not guaranteed that he can repudiate a document that he in fact did not sign.

Under American law, the burden of proving that Alex did sign is normally on Bob. This burden motivates Bob, at the outset of the transaction, to seek evidence of Alex's responsibility from things other than simply Alex's signature. This may mean that Bob would ask Alex to acknowledge his signature before a notary. More commonly, it means that Bob establishes a relationship with Alex in which they exchange feedback between each other: Bob asks for partial advance payment, Bob sends acknowledgments to Alex's independently verified address, or the like. The feedback reduces the risks to Bob.<3>

The myriad risks with a paper and ink signing are dis-

tributed across a number of features of the signing ritual: Alex's style of signing, Alex's secret choice whether to use his usual signature, the content of the signed document, the facts external to the document (such as any interaction between Alex and Bob) that place it in a historical context, the competence of the person who opines on the authenticity of the signature, and so on. In other words, the eggs are spread into many baskets. No single egg is highly reliable or highly important.

In a dispute over the authenticity of a document, the fact finder (the jury) does not look at the signature in a vacuum. Rather it considers all the relevant facts and circumstances -- the historical context of the document and all the ambient clues (such as corroborating records or testimony) that might bear on the authenticity question.

Just as risks plague the authentication of paper documents, so they will plague the authentication of electronic documents. To expect perfect binding of an individual like Alex to the words of an electronic document is not realistic.

To bind Alex to his electronic words, inventors might craft any number of strategies. One that has attracted attention is embodied in the Digital Signature Act adopted by the Utah Legislature in 1995 (Utah Act), which is codified at Utah Code Annotated Section 46-3-101 et seq. It contemplates that public-key cryptography would be used in the context of a large, global network of government-licensed certification authorities (CAs).

#### PUBLIC-KEY CRYPTOGRAPHY DESCRIBED

Public-key cryptography provides a mathematical scheme for arranging computer data (e.g., an electronic expense voucher or medical record) such that its integrity and origin can be proven.<sup><4></sup> Public-key cryptography involves the use of two keys (special strings of data), a private key and a public key, which are assigned to a user (Alex). Each key bears a complex mathematical relationship to the other. As the name suggests, the private key is intended to be kept secret, so that only Alex can access it. The public key, however, is not intended to be kept secret. It can be published so that outsiders can know it.

Suppose, for example, that Alex wants to sign a document for Bob in a way that confirms the document was really signed by Alex. After viewing the document's content, Alex would use his private key and a crypto program to attach ■ "digital signature" to the document.

The digital signature is a short unit of data that bears a mathematical relationship to the data in the document's content. Bob then could confirm the document's authenticity by using Alex's public key and a crypto program. If the document was not altered between the time Alex signed it and the time Bob confirmed, then the program informs Bob that the document was signed by someone possessing Alex's private key. Bob might infer then that Alex did sign the document. If the program cannot inform that the document was signed with Alex's private key, then Bob infers either that the document was not signed with Alex's private key or that it was changed after it was signed.

Public-key cryptography can be used in endlessly creative ways. The Utah Act contemplates using it in one particular way, what this article calls the "Utah Strategy." (The Utah Act is an admirable intellectual work; within limits it may support other strategies.)

#### THE UTAH STRATEGY

One of the risks in signing with public-key cryptography is that the person using the public/private key pair might not be the right one. A person claiming to be Alex may in fact not be Alex. To alleviate this risk, the Utah Strategy imposes an elaborate scheme for binding Alex to a particular public/private key pair.

First, a CA would be licensed by state government to ascertain the identities of people like Alex and link them to their key pairs. When the CA confirms Alex's identity and his control of the requisite key pair, it must ask him to use a new "distinguishing name," a computer code that labels Alex uniquely in all the world, so that the nominal link between Alex and his key pair is unmistakable. After the CA confirms Alex's identity, distinguishing name, and key pair, it issues a certificate affirming that Alex's distinguishing name is associated with the public half of the key pair. (The certificate must expire within three years, which implies that Alex must re-register with the CA regularly.) Alex is then obligated by the Utah Act not to allow his private key to fall into the hands of someone else. If Alex neglects his obligation, it would be difficult for him to avoid responsibility for documents signed with the key (e.g., demands for the withdrawal of money from his bank account), even if he did not approve them.

The Utah Strategy entails the keeping of secrets. Alex must keep his private key secret. But Alex will not be able to remember his key, so he will have to store it on a computer device such as a smart card and then keep the device in a safe place. In turn, the vendors of the smart

card and its various components will need to keep secrets. If the vendors fully disclose the methods they use to control the private key, then it will be easier for an attacker to steal the key.

Strictly speaking, public-key cryptography does not reduce risk in the signing of an electronic document. It transfers risk. It can be very effective in showing whether a particular document was signed with a certain private key.

But this transfer of risk does not necessarily result in the elimination or even the reduction of risk. Risk simply shifts onto the private key. That key becomes the object of any criminals who want to cheat Alex or Bob. They will try tricking Alex into revealing the key or temporarily surrendering control of it. They will endeavor to compromise the software that controls the key and its functions. Or they will steal and unlock the device, such as a smart card, in which Alex stores his key. If under the Utah Strategy millions of people were issued smart cards containing valuable private keys, then an underground industry of criminals would devote itself to corrupting those cards and the infrastructure that underpins them.

Not only does the Utah Strategy shift risk to the private key, it concentrates the risk there. The Utah Act gives recipients like Bob strong reason to expect that if a document is signed with Alex's private key then Alex is legally responsible for the document. Utah Act Section 46-3-401 provides that a document signed with a digital signature is normally presumed to be signed by the person owning the relevant private key (so long as his public key is certified by a licensed CA). This presumption reduces Bob's incentive to gather or consider any evidence other than the digital signature when he evaluates whether he can prove that Alex is responsible for a document. It allows Bob to forego the trouble of establishing a relationship with Alex or exchanging acknowledgements or other feedback with him to confirm his responsibility.

This presumption in turn gives Alex powerful incentive to protect the key. The incentive is much greater than the incentive consumers have to protect their automated teller machine cards. (Under the Electronic Funds Transfer Act, 15 U.S.C. 1693-1693r, consumer liability for a stolen ATM card is often limited to just \$50.)

Under the Utah Strategy, control of Alex's private key becomes all important. In other words, virtually all the eggs are placed in one basket — the private key (and the technology such as smart cards that protect that key

from villains). This allocation of risk may make sense for some transactions, particularly high-end financial deals initiated by sophisticated people. But a common person like Alex may not feel comfortable with it. He may not like:

1. dealing with a bureaucratic CA,
2. associating himself with a computerized distinguishing name, or
3. having responsibility for an extremely powerful private key.

Fortunately, the Utah Strategy is not the only way to allocate risk in the signing of electronic documents. Alternative strategies would spread the eggs among many baskets.

## PENOP

One strategy for spreading risk would use PenOp(tm).

### PENOP DESCRIBED

PenOp employs a pen biometric technology. It is a computer software component that augments the function of other computer applications, such as applications that control electronic documents. PenOp has two primary features:

1. The Signature Capture Service \*(SCS)\* captures and permits the storage of certain data associated with the manual inscription of a signature (autograph) on the screen of a pen-based computer (or a digitizer pad on a PC). The SCS must work with a "Client Application," which is software that informs the pen computer user what he is doing and prompts him when and how to do it. A Client Application can be designed to manage an electronic document such as an expense voucher.

In coordination with the Client Application, the SCS receives information, such as a user ID or a name, showing who the user (Alex) claims to be. It then prompts Alex to inscribe his signature, using a stylus (or pen), to a window on the computer's screen. It supplies the wording of the prompt in the window, known as the "Gravity Prompt," which indicates the purpose for which the signature is being captured. The Gravity Prompt normally refers to an electronic document that is accessible to Alex through the pen computer.

As Alex moves the stylus across the screen, an image appears that traces the movement of the stylus. Thus he sees his autograph. At the same time, the SCS measures certain features of the inscription event, including the size, shape, and relative positioning of the curves, loops, lines, dots, crosses and other features of the signature

being inscribed, as well as the relative speed at which each feature is imparted. The results of these measurements are known as "act-of-signing statistics." Alex then has the option, by tapping indicated buttons on the screen, of approving the inscription event, retrying it, or aborting it.

If Alex taps the approval button, the SCS calculates a checksum, or a brief string of data, that represents the content of the electronic document referred to by the Gravity Prompt. This checksum is not a complete statement of the original document, and the original document cannot be derived from the checksum. But the checksum bears a mathematical relationship to the document. If the document is changed, then it can no longer be mathematically matched with the checksum. (PenOp creates checksums using the MD5 digest algorithm by RSA Data Security, Inc.)

Next, the SCS compiles the following data (the "itemized data") and computes a second checksum from it:

- \* the first checksum
- \* the act-of-signing statistics
- \* the date and time of signing (as represented by the computer operating system under which the SCS is operating)
- \* the identity of the particular machine on which the signing occurred (based on identity information programmed earlier in the SCS)
- \* the claimed ID of the user (Alex)
- \* the words that appeared in the Gravity Prompt
- \* (optionally) data reflecting the graphic image of user's signature

The SCS creates the second checksum in two steps. First, the SCS retrieves a secret key previously programmed into the Client Application and uses that key to encrypt the itemized data. (This is the "first level of encryption.") Second, the SCS calculates from that encrypted data the second checksum. The second checksum establishes a link between the itemized data and the Client Application.

Finally the SCS encrypts the itemized data, plus the second checksum, using a different algorithm, one which does not use a secret key from the Client Application. This is the "second level of encryption". The resulting encrypted string of data – called the "Biometric Token" -- is a tamper-resistant representation of the event in which Alex inscribed his autograph.

2. The Signature Verification Service \*(SVS)\* reports the probability that a particular signature is authentic. First, in authorized enrollment sessions, the SCS cap-

tures and the SVS holds, in a database, act-of-signing statistics for a user like Alex who has been identified to the SVS.

Later, the SVS may be presented with a particular Biometric Token and directed to evaluate whether it is a product of an authentic inscription of the signature belonging to the user identified in the token. The service decrypts the token and then compares the information therein with the signature statistics stored earlier in its database. Based on this comparison, it issues a "signature match percentage", e.g., 50 percent or 72 percent, and reports this percentage to a Client Application (software configured to make use of the report). The SVS applies scientific principles deemed relevant by PenOp's developers.

#### PENOP STRATEGY

PenOp might be used to "sign" electronic documents such as contracts, expense reports, or medical records. Here is one strategy for doing so, what this article will call the "PenOp Strategy." Under this strategy, Bob seeks merely to have Alex attach a Biometric Token to an electronic document for the purpose of "signing" it. Bob does not seek to verify the Biometric Token at the time he receives it, just as he would not verify Alex's autograph at the time he receives paper from Alex.

To start, a Client Application within a pen computer is configured to display to Alex the data within the document in question (text, graphics, and so on, all in digital format). The Client Application then calls the SCS to write a Gravity Prompt, inviting Alex to "sign" the document by inscribing his signature within a window on the computer screen. The SCS also presents Alex a button for approving the inscription. If he inscribes and approves, SCS captures the necessary data and creates a Biometric Token. The SCS delivers the token to the Client Application for storage in a way that identifies the token as being related to the signed document.

If, at a later date, a third party such as a court wishes to verify that Alex did "sign" the document, it could obtain the PenOp SVS, introduce Alex to it, and use it with the help of an expert to verify (to the degree possible) that the Biometric Token represents an inscription by Alex.<5> A test could also be made (using the checksums in the Biometric Token) to establish whether the document to which the token is linked is the exact document used at the time of the token's creation. Another test could be made to confirm that the token was made with the identified Client Application.

The PenOp Strategy is similar to the traditional paper and ink strategy. Direct signature verification occurs only on rare occasions, and the full burden of proving that Alex signed a document rests with Bob. Bob is therefore motivated not to rely greatly on the signature; he will want to get evidence and security from other sources, such as advance payment or feedback from Alex.

## RISK ALLOCATION WITH PENOP

Like the traditional paper and ink strategy, the PenOp Strategy allocates risk across multiple factors (spreads the eggs to many baskets).

The creation of a biometric token that falsely appears to come from Alex requires the attacker to defeat security features (break eggs) that are supplied by these three different people:

1. The developer of the Client Application
2. The developer of PenOp
3. Alex

1. A Biometric Token must be made with the aid of an identifiable Client Application. The Client Application supplies the secret key that is used in the “first level of encryption.” If an attacker stole many proprietary secrets from PenOp’s developer he could learn in the abstract how to create false biometric tokens (see below).<sup><6></sup> But to create a false token that convincingly links a specific transaction to Alex, the attacker would also need to steal the secret key and other information from the developer of the Client Application. The degree and character of security that the Client Application employs is to be decided by its developer. That developer may, if it chooses, employ multiple secrets that are spread among multiple segregated parties (thus distributing more eggs into more baskets). It can employ audit trails, secure timestamps, physical access barriers, and even public-key cryptography as parts of its security (the Client’s secret key might be the private half of a public/private key pair). The greater the security, then the greater the forensic value of documents signed through that Client Application.

Before Bob relies very much on a Biometric Token, he will want to know about the reliability of the Client Application that helped create it. (Bob would have a similar concern if he were relying on a document signed by Alex with private key cryptography. The reliability of the link between a public key and a document signed by that key depends on the reliability of the software that controls the key and exposes it to the document it signs.)

2. The PenOp software employs a complex array of secrets that are difficult for an attacker to obtain, understand, and use. First, the methods PenOp uses to measure and record an act of signing are known only to PenOp’s developer, and those methods can change over time. Second, for the “second level of encryption” PenOp uses a novel encryption method that achieves these goals:

- \* Neither the SCS nor the SVS software possesses the key that encrypts a token at the second level of encryption.
- \* Neither the developer of PenOp nor the developer of the Client Application possesses the key that is used for the second level of encryption.
- \* Someone possessing the object code to the PenOp SVS software could decrypt a biometric token (at the second level of encryption) for purposes of a preliminary analysis of the checksums (linking the token to the original document) and the signature statistics (linking the token to Alex), but that person would not have the information to falsify tokens or inconspicuously corrupt them. A deeper analysis of the signature statistics would require expert advice and access to secrets kept by PenOp’s developer.
- \* To understand and replicate the second level of encryption, an attacker would need to steal the source code to the SCS and the SVS. The developer of PenOp intends to keep the source code a secret.
- \* A very sophisticated and determined attacker might be able break the second level of encryption. But the breaking of that level allows the attacker to overcome just one hurdle (break just one egg) in his fraudulent effort to create a signed document. What is more, the second level of encryption is highly resistant to known plaintext attacks.<sup><7></sup>

3. To create a convincing biometric token purporting to be from Alex, an attacker would need to obtain extensive information about Alex’s signing behavior. For the attacker, this would be a considerable burden.

Even if an attacker successfully tricks the developer of a responsible Client Application, the developer of PenOp, and Alex into disclosing the necessary information, the attacker would still have much work ahead of him (more eggs to break). To perpetrate a fraud, he would have to fabricate a convincing transaction, one that makes sense under the prevailing facts and circumstances. It would have to be consistent with the types of transactions Alex would have entered at the time, including the records that Alex himself and other interested parties would

keep. If, for example, Alex were a school teacher of modest means, a corporate debenture that appears to bear his signature would not be convincing.<sup><8></sup> Bob would have a hard time carrying his burden of proof that Alex signed that document.

## AUTHENTICATION IS AN ENDLESS JOURNEY

No particular application of paper and ink, the Utah Strategy, or the PenOp Strategy can provide a perfect bond between Alex and the words of a document. The development and use of authentication technology is a dynamic process. It is not a destination; it is an endless journey in which the good people hurry to stay a step or two ahead of the bad people. The paper and ink tools that provided adequate authentication in the year 1900 do not necessarily provide the same level of authentication in the year 1995 (consider how the sophistication of money counterfeiters has grown in the past 95 years). The computer security tools that provided adequate authentication in 1970 do not provide the same level of authentication in 1995.

Similarly, the tools needed to provide adequate protection for a private crypto key in 1995 will be different from those needed to provide equivalent protection in 2010. And, yes, the tools needed to protect a PenOp application from abuse in 1995 will be different from those needed in 2010. As PenOp becomes more popular, PenOp's developer may need to divide and spread PenOp's secrets (some of its eggs) across a larger number of people. Under both the Utah Strategy and the PenOp Strategy, systems developers will have to work endlessly to keep their secrets out of the hands of criminals.

In principle, neither the Utah nor the PenOp Strategy is an inherently superior method for connecting Alex to his electronic words. Each is an approach for staying one step ahead of the bad people. Whether any particular application of these strategies is adequate will depend on all the facts and circumstances of a particular transaction.

The chief difference between the strategies lies in the ways that people will use them. The Utah Strategy emphasizes the investment of many eggs in one basket - the private key; whereas the PenOp Strategy (like the old paper and ink strategy) emphasizes the spreading of eggs across many baskets.

## POPULAR ELECTRONIC COMMERCE

What does this difference in risk spreading mean? It

means that the Utah Strategy may not appeal so much to members of the general public.

The Utah Strategy stresses the responsibility of Alex to protect his private key, and places a light burden of proof on Bob. In contrast, the PenOp Strategy, like the traditional paper strategy, stresses that Alex and Bob act reasonably under the circumstances. Evidence of signing comes from all the relevant facts, with the full burden of proof being on Bob.

The Utah Strategy relies on the creation of a complex network of CAs, and requires Alex to register his identity with a CA (repeating the registration every three years) and to take a new distinguishing name (a socially and politically sensitive requirement). The PenOp Strategy, on the other hand, requires no advance planning or action on Alex's part. It caters to Alex's interests, and his instincts for how authentication should work, thus making electronic commerce attractive to consumers and common people.

## ENDNOTES

<1> This article does not consider whether any given method will be considered in law as a signature. For more on that topic, see Benjamin Wright, *The Law of Electronic Commerce*.

<2> In practice, disputes over the authenticity of commercial documents are rare. Of the many billions of commercial documents created every year, the authenticity of only a tiny fraction of the total is seriously contested in court. Among the reasons for this are that most people are happy with their commercial transactions most of the time and the facts and circumstances surrounding the documents (including the signatures, but also including the context and content of the documents) tend to show their origin and authenticity.

<3> See "Legal Identity and Signatures on the Information Highway," a component to Benjamin Wright, *The Law of Electronic Commerce*.

<4> Public-key cryptography is but one of many tools in the world of cryptography. Even within the field of public-key cryptography, there are many specific technologies. But for understanding the issues discussed here, it is adequate just to consider public key as a single, generic technology. Depending on how implemented, various public key technologies can perform some or all of the functions described here. Cryptography is a complex topic. Any reader who seeks to achieve certain results from cryptography should consult a competent professional. For more on cryptography, see Bruce Schneier, *Applied Cryptography* (John

Wiley and Sons) 1994.

<5> Alex's cooperation, although helpful, would not necessarily be required at the time his signature is verified. According to PenOp's developer, even if Alex is dead or refuses to cooperate, a limited forensic comparison could still be made between his signature, as captured earlier by PenOp, and one or more specimens of this signature as written on sundry paper documents such as checks, letters, contracts, or credit card receipts.

<6> As the PenOp Strategy is implemented, secret information will need to be divided and spread among segregated parties. Similarly, as the Utah Strategy is implemented, secret information about the private keys and their security (e.g., information about the function and control of smart cards and supporting software) will have to be divided and spread among segregated parties.

<7> A "known plaintext attack" is one in which the attacker treats the encryption algorithm as a black box. He takes a piece of known plaintext (e.g., a string of zeros), selects a key, and tries to discern how the algorithm works by running the text and key through the algorithm. He then analyzes the resulting encrypted text. For example, when a string of zeros is run with the key "KEY" through a very simple algorithm, the encrypted result might be "KEYKEYKEY".

But PenOp's second level of encryption is highly resistant to a known plaintext attack because it never encrypts the same block of data twice in the same way. The reason is that it uses a random encryption key.

<8> As the Information Age progresses, records about transactions become far more extensive and detailed than was true before. And the records become spread among many (and sometimes unexpected) parties, including sundry network service providers. The massive audit trail that will build up around commerce will make for an environment in which fraud is more difficult, not less. More of the facts and circumstances that surround a transaction will be recorded by independent third parties and other reliable means.

## II SIGNING TAX RETURNS WITH A DIGITAL PEN

Until now, when a taxpayer files a tax return with the Internal Revenue Service electronically, it has been required that she sign a separate paper form, form 8453, as an authentication of the return. The paper 8453 form, which summarizes key parts of the tax return, then must

be mailed to the IRS. The reason for the paper is that there was no electronic authentication device as effective and convenient for taxpayers as the ink signature. Now, in a bid to eliminate paper entirely, the IRS is accepting electronic authentications for some returns.

### HOW THE SYSTEM WORKS

After a tax preparer (like H&R Block) assembles a tax return for electronic transmission to the IRS, the preparer displays to the taxpayer an electronic 8453 form on a computer monitor. The taxpayer then authenticates the form by picking up a pen or stylus and signing her autograph on a digitizer pad attached to the computer.

The digitizer pad coordinates with the PenOp(r) software, which captures a bitmap image of the signature (somewhat like an image on a fax) as well as statistical measurements of the signing event. These statistical measurements, known as biometric measurements, record the unique behavior that any individual exhibits when signing his autograph, such as the speed, angle, and grace with which the various loops, lines, and crosses of the autograph are imparted.

PenOp is the digital descendant of ink on paper, and it aims to do the same thing ink does. PenOp's developer maintains that the biometric measurements, in combination with the captured bitmap, store evidence of the signature similar to the evidence stored by marks on paper. In other words, the developer believes it can demonstrate that a qualified handwriting expert can be just as effective judging the authenticity of a signature captured with PenOp as he would be judging the same signature inscribed in ink on paper. Handwriting analysis, of course, is an imperfect art, and there is no guarantee PenOp's measurements will be accepted as credible evidence in court, just as there is no guarantee that any given ink-on-paper signature will be accepted.

When the taxpayer signs with PenOp, the biometric measurements are cryptographically united with a hash summary (or mathematical digest) of the 8453 form, thus forming a "biometric token" which is transmitted along with the form to the IRS. (*For white papers and other information on PenOp, see [www.penop.com](http://www.penop.com).*)

When the Service initially receives the electronic return, it does not open the biometric token or verify the signature information in it. Typically, it will never try specifically to verify the signature. (Similarly, few if any signatures on paper-and-ink documents filed with government agencies — tax authorities, courts, county registries, etc. — are verified upon receipt. Verification of ink signatures is not worth the effort. Sometime in the

future, however, agencies such as IRS might try to use technologies like PenOp to verify electronic signatures upon receipt.)

The electronic signing with PenOp occurs within a controlled environment. The tax preparer is registered and regulated by the IRS, and subject to severe criminal and other penalties for abuse of the system. The preparer is identified on the 8453 form when it is submitted to the IRS, which occurs through a proprietary system open only to authorized tax preparers. When the signing process is complete, the preparer is required to hand the taxpayer a paper record of her return and 8453; and if the preparer neglects to offer the paper record, the taxpayer intuitively knows to ask for it because she needs something to keep with all her other tax records.

In the event of a later dispute -- in which the taxpayer takes the bold step of specifically repudiating her return -- it is most likely the IRS would prove the return's authenticity by reference, not so much to the signature, but rather to (1) the private information (such as employer withholding data) in the return, (2) the exchange of money (such as a payment or a refund) or other documents (such as correspondence with auditors) in relation to the return, or (3) records retained by the tax preparer or the taxpayer herself. Only in very rare disputes would the IRS have to rely greatly on the bitmap and biometric measurements buried in the PenOp biometric token.

#### WHAT DOES PENOP CONTRIBUTE?

Given that the IRS does not rely heavily on the PenOp-captured signature to prevent repudiation, what purpose does it really serve? The answer is threefold.

**FIRST**, the IRS does check signatures on a fair number tax returns by simply asking taxpayers (such as at time of audit) whether the signatures on the returns are theirs. Of course taxpayers can deny their signatures if they choose, but normally they will not. (Reasons: normally the signatures are indeed theirs, normally taxpayers are honest, and normally taxpayers know that the origin of their returns can be proven from the combination of the signatures and the other facts and circumstances surrounding the returns.) This is an important, practical feature of the handwritten signature, and PenOp supports it. PenOp binds the 8453 form to a signature image, which when presented to the taxpayer later can help her verify the 8453 by visual inspection.

**SECOND**, the IRS needs to have the taxpayer do something that actively shows she believes the tax return is

complete and she assents to it. Physical action by the taxpayer separates a draft document, which has no legal effect, from a final one, which is legally binding. The Service wants something that replaces the simplicity, intuitive meaning, modest reliability, and low cost of an autograph on paper.

This need for physical action might be satisfied by any number of methods, including PenOp, voiceprint capture, or the capture of a simple bitmap signature (United Parcel Service delivery people have been using the bitmap method for several years). The disadvantage of a bitmap by itself is that it is too easy for the tax preparer (e.g., H&R Block employee) to fabricate. A bitmap is just a simple scanned image of a graphic like an autograph. If the tax preparer has access to a bank check, a credit card receipt, or prior tax return signed by the taxpayer, it wouldn't be extremely difficult for him inconspicuously to create a false tax return. What is more, even if the tax preparer did not have a sample signature, he might be able to make a passable bitmap just by guessing what it would look like.

PenOp addresses this disadvantage. In addition to capturing a bitmap, PenOp captures statistics about the dynamic process of handwriting an autograph. The statistics are unique evidence linking the signer to the signing event roughly similar to the way an ink and paper signature does. The method PenOp uses to calculate and store those statistics is a complex secret hidden from the tax preparer. The secret can be changed from time to time by PenOp's developer and by the IRS, outside the knowledge of the preparer. PenOp therefore is a barrier to any tax preparer aiming to fabricate ■ return.

**THIRD**, PenOp deters a taxpayer from repudiating her tax return, and it does so in a way similar to the paper-and-ink signature. Suppose that a taxpayer signs an electronic 8453 form and receives from the tax preparer a paper record of it. Later, she deceitfully tries to repudiate the 8453. She claims that she did give her tax information to the preparer (and thus the 8453 form and the corresponding return report information known only to her), but that she never finally agreed to or signed the 8453 form in question. She claims she never received a paper record of the 8453 form, and the signature on the electronic 8453 held by the IRS was forged by the tax preparer.

An investigation could well reveal how hollow her claims are, if she signed with PenOp using her normal autograph. It would have required great effort for the tax preparer, or most anyone else, to forge her signature, for he does not know or have access to the biometric measurements of her signature (even if he does have a

static copy or bitmap of the image of her signature). To have obtained the measurements by ruse would have been a considerable challenge.

What if the taxpayer had signed with PenOp, but had deliberately elected not to use her "normal" autograph? What if, instead of signing her autograph, she sketched a picture of a bird or a flower? Might she stand a better chance at repudiating her signing of the return? The answer might be yes. However, this risk of repudiation of a PenOp signature is equivalent to a risk that has always existed with paper tax returns. Taxpayers are free to use strange marks on their tax returns as their signatures. Paper and ink don't prevent this risk, and neither does PenOp. PenOp strives to offer the same risks and benefits as paper and ink.

## DETERRENCE TO FRAUD

PenOp in and of itself does not prevent all the frauds a tax preparer might attempt with an electronic tax filing. PenOp by itself does not, for example, absolutely prevent the preparer from breaking into the application software that controls the 8453 form (software supplied by the IRS) and causing it to mislead PenOp as to which document is being signed or causing it to show on the computer monitor information that is different from what is being signed. However, these frauds are deterred, among other things, by two factors:

- (1) The taxpayer intuitively expects to receive a paper record of the return and the 8453. If the paper record does not match with what the tax preparer sends to the IRS, then the preparer (who is known and regulated by the IRS) risks being caught and punished.
- (2) Tinkering with the IRS's application software involves considerable work.

In assessing this deterrence, notice that the malicious tax preparer has little incentive to undertake much effort. The falsification of a PenOp signature would not win him a quick refund from the IRS. The IRS does not rely on signatures (whether ink/paper or PenOp) to guard at time of receipt against false requests for refunds. Signatures are merely \*latent\* controls that serve only in the event of relatively rare and expensive investigations of tax returns. Rather, to catch false refund requests the Service relies on \*immediate\* controls, such as independently verified information, such as tax withholding reports from the taxpayer's employer.

## A REPLACEMENT FOR INK

PenOp's mission is simple: it aims to compete with and

replace its analog ancestor, the ink signature on paper. Like its ancestor, the PenOp signature intuitively informs the taxpayer that she is legally binding herself to her 8453 form. Subsequently, it helps her verify her return by visual inspection. And, like the properties of paper and ink, the biometric measurements in PenOp make it more difficult for the taxpayer who signed an 8453 later to repudiate it claiming the signature was forged by the tax preparer.

*Benjamin Wright (73457.2362@compuserve.com), a Dallas-based attorney, is special counsel to PenOp, Inc. ([www.penop.com](http://www.penop.com); tel: (800) 286-4137). He is also author of The Law of Electronic Commerce: EDI, Fax and E-mail, published by Little, Brown and Company (tel: (800) 331-1664; fax: +1-617-890-0875).*

*Mr. Wright is not an engineer, a computer scientist, or a forensics expert. He has assumed the technologies identified here function and are implemented in competent, reliable, and credible ways and adequate records are maintained. No warranty is given as to the accuracy or completeness of the information in this article. The transaction of commerce is inherently risky, and nothing published by Wright advises which level of risk is appropriate for you.*

*For more articles on the legality of electronic commerce, see [http://infohaus.com/access/by-seller/Benjamin\\_Wright](http://infohaus.com/access/by-seller/Benjamin_Wright).*

*Copyright Benjamin Wright. This article may be copied, reprinted and redistributed so long as it is not modified and no parts (including the four paragraphs of this notice) are removed.*



# Digital Currency and Public Networks: so what if it is secure, is it money?

John du Pré Gauntt

*Research Associate*

*Department of Information Systems  
The London School of Economics  
Houghton Street  
London WC2A 2AE  
tel 44-171-955-7638 (Prof. Ian Angell)  
email ag112120@dircon.co.uk*

*Business Editor*

*Public Network Europe  
The Economist Group  
15 Regent Street  
London SW1Y 4LR  
44-171-830-1061  
fax 44-171-839-1475*

## Abstract

This paper will comment on some of the issues concerning interactive payment systems over the PSTN. Holding as axiomatic that widespread value transactions could evolve telecommunications networks into public electronic markets, the author will explore some of the ramifications. Networks as markets imply that communications resources could become a volatile commodity, and it is suggested that certain market participants will take greater control over the factors vital for their commercial well being.

Among the most influential actors could be the payment systems providers themselves who must construct an asset portfolio to back their digital currency. Moreover, such a portfolio must include assets which can be redeemed in real time and which are desired throughout the electronic market. It is suggested that a feasible asset for backing digital currency would be the infrastructure which makes it possible in the first place. In essence, the telecommunications network will back the money circulating in an electronic market.

It may turn out that payment service providers will buy large chunks of telecommunications capacity or monetise the traffic revenue streams to back their digital currency. This could change the telecommunications industry and finance in ways not yet imagined.

## Introduction

*Networking technologies can greatly reduce the costs entailed in exchange transactions. As these costs decline, many business activities will be shifted to the marketplace...The network will, in many instances, serve as the market. When this occurs, market structure will depend as much on network characteristics and the economies of networks as it does on relationships among firms.[1]*

This paragraph by the United States Congress' Office of Technology Assessment (OTA) leaves little doubt that policy makers are considering seriously the idea of public telecommunications networks serving as a primary infrastructure for electronically distributing goods and services.

Under the rubric of electronic commerce, telecommunications networks are being perceived as a prime enabler of a new intellectual property and information services based paradigm of economic production and distribution. This new model integrates communications, data management and security services that allow business applications within different organisations or households to automatically interchange information.[2] Perhaps the most important information exchanges will be transfers of value represented by digital media between buyers and sellers.

Among those hoping to use the public switched telecommunications network (PSTN) as a platform for offering financial services are established banking and credit card institutions. They are betting that internet-worked PC's could migrate electronic commerce—and demand for financial mediation—from the periphery to the centre of the consumer economy. The primary perceived value is replacement of the physical infrastructure with a network based infrastructure. This will enable financial service providers to scale back expensive brick and mortar branch systems while reaching

greater numbers of customers.

Another major application area within electronic commerce involves creating and distributing exchange instruments possessing embodied value in their own right. In this case, the operator is supplying an artefact which represents cash with sufficient rigor and completeness as to become a reliable surrogate for paper and coin currency.

Thus, financial services as a whole are employing the public networks as both a new *environment* for electronic commerce and as a distribution vehicle for a new *manifestation* of value. It is this latter aspect of electronic commerce—transactions involving digital currency—which concerns the body of this work.

Currently, value exchanges employing digital currency are rare and have not picked up even though the public has been offered a plethora of industry and government predictions about the coming flood of electronic money. The reason most often cited has been the lack of a standard, secure protocol for transacting value over the PSTN.

But this is not a convincing argument. Privacy issues notwithstanding, there is no conceptual reason why technically proficient cryptographic transaction systems won't be available commercially in the near future. The writer believes that the current security debates concerning digital currency have obscured the more important question of the extent to which these products are to be considered 'money' in the economic and political sense of the word.

Lacking the status of legal tender—a right unlikely to be given up by the central banks anytime soon—the acceptance of digital currency will have to be based upon the public's trust that not only is it secure but that it is valuable. This will be determined by the market's confidence but influenced by the asset base with which an issuer backs their currency.

This paper will take cryptographically secure digital currency as given. Thus, the primary issue will be how the determination of an asset portfolio to support digital currency could alter traditional notions of finance and the telecommunications industry.

This paper is divided into four sections. The first section will give selected views from the literature concerning electronic markets and will define telecommunications as a volatile resource. The second section will deal with some of the requirements for a real-time electronic currency circulating in a networked market. Section three will propose capitalisation of the network infrastructure or telecommunications traffic streams into commodity contracts which could back digital currency. This may lead to the situation of currency issuers becoming among the largest non-teleo buyers and resellers of bandwidth capacity. The last section will

comment on some of the possible effects caused by the evolution of public networks toward electronic markets.

## Electronic Markets and Volatility

Certain writers suggest that economies have two primary mechanisms for co-ordinating the flow of materials and services through adjacent steps in the value chain—markets and hierarchies.[3] Williamson categorises those transactions that support co-ordination between multiple buyers and multiple sellers as market transactions and those which support co-ordination within the firm or group of firms as hierarchical transactions.[4] Whether a market or a hierarchical transaction, a primary goal of firms is to manage the costs of distributing goods or services.

Malone et al classify costs as falling into the categories of production and co-ordination costs. Production costs include the physical or other primary processes necessary to create goods and services while co-ordination costs entail all the information processing necessary for their distribution.[5]

According to this model, markets have been characterised by low production costs and high co-ordination costs while hierarchies display the reverse. It follows that for market transactions, firms should prefer transaction bundles which economise on co-ordination costs and it is here where information technology and telecommunications have an important role to play.

A major inefficiency involved in market transactions relates to search costs incurred by buyers obtaining information about price and product offerings of sellers.[6] One of the main promises of electronic commerce will be the ability of firms to reach masses of customers and vice-versa through the public networks. Platforms such as the World Wide Web (WWW) owe much of their popularity to the ease with which users can 'surf' around the world, moving in and out of web sites with the click of a mouse. This has lowered many of the costs associated with finding out product or service information and is the inspiration for many firms setting up a presence on the Web.

But this is not yet a market. Bakos defines an electronic marketplace or electronic market system as an interorganizational information system that allows participants—whether buyers or sellers—to exchange information about price or product attributes.[7] He specifies five economic characteristics that distinguish this mode of exchange:

- a.) An electronic market system can reduce customer's costs of obtaining information about prices and product offering of alternative suppliers as well as suppliers'

costs of communicating information about their prices and product characteristics to additional customers.

- b.) The benefits realised by participants in an electronic market place increase as more organisations join the system.
- c.) Electronic marketplaces can impose significant switching costs on their participants.
- d.) Electronic marketplaces typically require large capital investments and offer substantial economies of scale and scope.
- e.) Potential participants in electronic market places face substantial uncertainties regarding the actual benefits of joining such a system. Occasionally this uncertainty remains even after an organisation joins the system.[8]

If we combine aspects of the markets/hierarchy model and electronic marketplaces; the costs to sellers falls in Malone's world while in Bakos' world, it is the search costs of buyers. If this is joined to a system which can exchange value rapidly, securely and effectively for both, a virtuous circle of buyers with low search costs transacting with sellers enjoying low distribution costs could enhance the attraction of electronic markets as a macro-level platform for distributing goods and services. If it is true that the National Information Infrastructure (NII) or other initiatives of this type will reduce dramatically the costs of information and communications and thereby accelerate the evolution of electronic market systems, it follows that a fundamental restructure of opportunities and roles within traditional industries and payment service providers will result.[9]

It is imaginable that many firms may perceive efficiency gains within electronic markets and therefore conduct more of their transactions on-line. A critical mass of firms or households might start from the proposition that they are going to manage the majority of their transactions over networks. Yet this creates new risks while solving old problems.

Electronic markets promise to lower co-ordination costs but at the price of increasing participants' exposure to the performance characteristics of the telecommunications platform. This comes about as more and more firms and/or households tie their economic well being to ease of access to secure and efficient telecommunications resources on a continuous basis and often from multiple locations.

The nineteen hour shut down of America On Line (AOL) due to software glitches brought home to many network planners the chasm between operational challenges of high capacity interactive networks and public expectations as to their reliability. Between 4am and 10:45pm on Wednesday August 7, 1996, up to 6.2million AOL subscribers were unable to use the service.[10] During routine maintenance procedures, AOL installed new software and replaced switches in the internal network of the main data centre. However, a faulty change in routing instructions meant that attempts to restart the system failed. Subsequent efforts to return to the original state made the situation go from bad to worse, culminating in an public apology by AOL chief executive Steve Case noting that it had a long way to go before it became as reliable as 'must-have' utilities such as electrical power and the telephone.[11]

On the other end of the spectrum are new risks that can arise from success. A firm setting up a transaction system to serve a base level of users exchanging value for a commodity is an understood problem. Imagine that the same firm has a hit product. Within a single business day, the estimated user population may increase by several orders of magnitude. This is a feasible yet not understood problem—how to scale and disassemble communications and information infrastructures over the short term.

Furthermore, information carried on these networks has become a factor of production in itself. Haeckel and Nolan hold that information's primary function is changing from recording what happens to making things happen.[12] Telecommunications and computer technology impact the value of information by processing it faster. Greater speed implies that more decision options can be explored in more depth in a given time.[13]

It follows that as more production and consumption decisions are predicated on the speed and reliability of interactive telecommunications infrastructure, the shift towards widespread commercial transactions over the public networks could redefine telecommunications as a volatile resource. If one remembers the role of fossil fuel energy towards the well being of industrial civilisation, one can imagine future perceptions of telecommunications platforms towards the prosperity of information civilisation.

Disruption, overload or sabotage can impact telecommunications dependent market systems as much as shocks to energy supplies affect manufacturing based market systems. Daniel Bell wrote in 1973 that codified information and knowledge are replacing capital and energy as the primary wealth creating assets, just as capital and energy replaced land and labour 200 years ago.[14] It would be extremely dangerous to assume

that the electronic platform which transforms and transports these assets is not connected decisively with their overall utility.

When one considers the nature of the service which is to be offered by payment providers—safe and efficient conveyance of value through the use of their instruments—one is referring to arguably the most risk sensitive enterprise on the network. An on-line bank or financial intermediary whose web site is technically unavailable or whose instrument's security or solvency is in question jeopardises its entire existence within a business day.

Thus, there exists for financial services the very real problem of creating and maintaining confidence in a public electronic market. Moreover, the new network model has lowered entry costs to the point where previous brand strength is an advantage, but not a decisive advantage over competitors arriving from different sectors who enjoy different cost structures and ways of looking at financial mediation.

It is highly unlikely with so much at stake that existing financial service providers will view telecommunications as merely something that they rent in order to hook up their information systems and offer services. If anything, information systems and telecommunications platforms are their capital stock and it is naive to presume that they will settle for anything less than significant control over a resource so essential to their well being.

### Money and its uses

Georg Simmel was a social philosopher rather than an economist who declared that "not a single line" of his most famous work was "meant to be a statement about economics."<sup>[15]</sup> For Simmel, the foundation of any monetary order is trust—a phenomenon often expressed in intrinsically useless fiat money, but which is true of commodity monies as well. It underpins markets and is facilitated by the desirability of money as a transaction medium and its ability to maintain value. Even the acceptance of metallic money:

cannot develop without public confidence in the issuing government, or perhaps in the real value of the coin in relation to its nominal value. The inscription on the coins of Malta—*non aes sed fides* [not metal but trust]—indicates very appropriately the element of trust without which even a coin of full value cannot perform its function in most cases.<sup>[16]</sup>

Money is often tied with the concept of legal tender but such arguments are based on politics rather than

practice. According to Simmel, legal tender laws appearing to ensure the acceptance of a particular money rely on the "probability that every individual, in spite of his ability to refuse the money, will accept it."<sup>[17]</sup> Von Mises' argument was similar:

The law may declare anything it likes to be [legal tender]. But bestowing the property of legal tender on a thing does not suffice to make it money in the economic sense. Goods can become common media of exchange only through the practice of those who take part in commercial transactions; and it is the valuations of those persons alone that determine the exchange ratios of the market. Quite possibly, commerce may take into use those things to which the state has ascribed the power of payment; but it need not do so. It may, if it likes, reject them.<sup>[18]</sup>

Writers addressing the history of money have regularly distinguished three types of currency: (1) that which owes its value, as do gold or silver, to an inherent desirability derived from pride of possession, prestige of ownership, personal adornment etc.; (2) that which can be readily exchanged for something of inherent desirability or which carries the promise of eventual exchange; and (3) currency which is intrinsically worthless, carries no promise that it will be redeemed in anything useful or desirable and which is sustained by the fiat of the state.<sup>[19]</sup>

It is unlikely that demand for digital currency will be based upon aesthetic appeal or any intrinsic desirability as per gold or silver, nor is it apparent that the nation-state is amenable to bestowing legal tender status on electronic cash. Thus, digital currency will have to draw its value based upon the portfolio for which it can be redeemed or the market's confidence in the issuer's promise of redemption.

Whether issuing currency for an electronic market should be the prerogative of national governments, banks or specialist transaction houses is not yet known. Fama describes banks as having two functions; the first being to provide transactions and the second being portfolio management.<sup>[20]</sup> Other writers concentrate on banks' role in the negotiation of credit through a loan of other people's money, and the issuance of fiduciary media, that is, notes and bank balances that are not covered by money.<sup>[21]</sup>

Regardless of specialisation, financial service providers must construct an asset portfolio which balances the exposure peculiar to their orientation and which allows them to leverage demand for their instruments. This demand will be influenced by the extent to which they insure the integrity of the exchange process

as well as their ability to handle the heightened velocity of electronic market transactions.

Regarding exchange, Camp et al. delineate the qualities pertinent to the mechanics of transactions under the Atomic, Consistent, Isolated and Durable (ACID) model.[22] Consider buying an item using some form of money at a place of business. First the transaction should be atomic. That is, the two events: 1.) the merchant receiving exchange-value, and 2.) the buyer receiving an artefact having use-value should be inextricably linked.

Second, the transaction results should be consistent. If the buyer thinks that they have given a token of value, the merchant should agree that they have received the token. It should never be the case where both buyer and merchant conclude that they possess the token or that they simultaneously believe that the other party has the token, in which case the token has disappeared! If the buyer believes that they have aborted the transaction, the merchant should also believe that the transaction is aborted.

Third, the transaction should be isolated. If, at the same visit, the buyer decides to purchase other commodities, the transactions should be clearly separable. Finally, the results should be durable. If there is some breakdown in the middle of the transaction—i.e. the buyer accidentally drops the token as they are handing it to the merchant—it should be possible to recover a consistent state—i.e. the buyer picks up the token and the transaction can start over.[23] Besides the ACID properties, the effort and cost associated with conducting a transaction should be sufficiently low as make large numbers of such exchanges feasible, not only in their frequency but also in the magnitudes of the values being exchanged.[24]

Existing alongside the challenges of establishing secure environments for transacting is the likelihood of increased velocity. The velocity concept attempted to explain why mercantilist Spain, with all its gold, produced an inflation rife society. The Spaniards would not have suffered ruinous upward price volatility according to the anti-mercantilist theories of Smith and Hume, had they fashioned the plundered gold from the New World into goblets or religious ornaments. The gold would not have then entered the money economy in competition for a limited amount of real goods.[25] Solomon argues that the same conditions hold for electronic money.[26] Moreover, the new money is able to turn over at electronic speeds.

One can extrapolate that if a digital currency exhibits ACID properties and experiences significantly increased velocity, the new electronic markets could alter conventional notions of money itself, prompting an observer to ask whether the tokens are being used are

artefacts which are inventoried at discrete points in time or processes which are not. Distinguishing between the two will be vital for coherent monetary policy.

Henderson suggests that money, reflecting historical practices, is expressed in an ever changing array of "M" definitions substantially bound to physical attributes.[27] Thus, money in theory remains an inventory. However, money in an electronic marketplace includes forms which are substantially free of physical limitations and—paced by electronic based transfers—money is becoming a flow and ■ process.

Current public policy issues that arise from money and payment processes reflect the reality that has prevailed in past decades. Money is conceived, defined and measured in physical terms; and payment processes are visualised, regulated, and evaluated in physical terms. Thus money came to be defined and measured when it came to 'rest' at the close of the business day.[28] Yet electronic markets will be by definition global and continuous consumer markets and the concept of the business 'day' will owe more to the parameters governing a particular market segment than some standard unit of time.

The earlier electronic funds transfer models used an electronic delivery mode but did not alter the backing for currency nor take electronic money out of the conventional M classifications.[29] Furthermore, this delivery mode existed for the large part between firms or across relatively closed networks. Granted that modern Automated Teller Machines (ATMs) allow individuals to deposit or withdraw money over the wide area and a smaller proportion allow for remote payment, these platforms are integrated into the member banks' settlement and risk management systems.

However, as electronic markets move towards some of the network-based transaction possibilities of the future (i.e. cross-border, multi-currency, bank and non-bank mediation), the fusion of old M classifications and new V classifications becomes more likely. Solomon notes that hypothetically, there may be no M or monetary base, merely continuous turnover in the money flows total.[30] But in this case, the money multiplier shifts up to infinity and becomes nonsensical.

Substituted by necessity would be ■ new payments standard based on commodities or electronic barter. In this scenario, instead of gold or government securities, there will be 'quasi-monetary' backing of derived securities in place of required reserves and could result in a financial service provider's general reputation or stores of commodities determining the value of the currency they issue for use in an electronic market.[31]

Thus, it can be argued that the perception of an electronic market of the value of a digital currency

which is useful as a transaction medium and which holds its value will be influenced by the desirability of an issuer's commodity or security bundle. The main question would be the degree of fungibility between the asset and the artefact it supports as well as whether traditional assets for backing money would be adequate for an electronic marketplace. This writer is agnostic that traditional securities offer a long term solution unless substantially altered or augmented.

Most textbook arguments state that the central bank controls the money supply through policy tools, the reserve base, and the interest rate. Through open market operations or loans at the discount window, central banks can create or destroy member bank reserves using government securities. The value of such securities is drawn from the state's right to audit and tax a national economy. However, the taxation regimes in most economies are ill-prepared to measure, let alone tax, intellectual property and information service based systems. Historically they are oriented towards modelling national economic performance—and therefore monetary policy—upon manufacture and consumption of physical commodities.

Further, current monetary policy assumes that much of the current digital currency will be backed by deposits of 'real' currency in bank accounts. Beyond whether this 'real' currency will be US dollars, German DM or whatever, there is the issue of how long the objective exchange values of electronic currencies will not be altered by changes in the exchange values of their national currency counterparts and vice versa. This would imply that attempts to 'freeze' bank deposits for backing electronic media could open many avenues for speculation on the spread between the two.

This suggests that traditional methods of controlling a national money supply in a multi-currency consumer electronic market would be even more indirect and would lengthen the time required for policy actions to be reflected in the market. It is likely, therefore, that any attempt to peg the value of a digital currency to the value of national currency will be short lived.

A digital currency based upon a basket of national currencies may have more promise yet experience in the European Union with trying to merge the monetary policies of multiple economies to arrive at a stable supranational currency—digital or not—leave little doubt that an electronic market would find it preferable not to transfer the problems experienced in political space into cyberspace.

Therefore, there is a need for constructing a portfolio which can be quantified and is desired by electronic market participants. Such a bundle may include national currencies but need not as per Von Mises' argument. What is required is the inclusion of an asset that

is useful, difficult to substitute and is the most marketable commodity in the electronic market.

This paper proposes that the telecommunications capacity which enables the electronic marketplace, henceforth referred to as bandwidth, could serve as a commodity which possesses many of the 'quasi-monetary' attributes needed for backing digital currency. In that kind of world, the telecommunications network backs the electronic market.

### Trading bandwidth

If we accept that the transaction model for intellectual property and service based economies will be electronic markets, and if we accept that telecommunications in that setting becomes a volatile resource, it is feasible that bandwidth could be capitalised and traded in a public setting.

There are two main reasons to trade bandwidth: (1) to hedge risk; and (2) to speculate on its price. Hedgers and speculators on either side not only create a new market, but could turn bandwidth contracts into assets to support digital currency. The primary issue would be whether one capitalises and trades the physical infrastructure or traffic flows on that infrastructure.

In the first scenario, it is possible for financial service providers to capitalise and trade aspects of telecommunications infrastructure without becoming operators themselves. This is the case of the Fibre optic Link Around the Globe (FLAG) submarine cable. The FLAG cable will be roughly 27,000km of two five gigabit fibre pairs which will link the United Kingdom to Japan along thirteen landing points.[32] Containing some 120,000 digital circuits operating at 64 Kbs, FLAG will allow for video conferencing, MANs, imaging as well as voice.

FLAG is unique in that the majority of its owners are not telecommunications operators but include investment houses such as Dallah Al-Baraka of Saudi Arabia, Marubeni of Japan, Thailand's Telecom Holdings and the Asian Infrastructure Fund.[33] American RBOC Nynex is the only telecommunications operator involved in FLAG. Nynex will provide project management along with being the managing sponsor with its 40% stake in the US\$1.2 billion project. The other partners in FLAG are investors from outside of the telecommunications sector.[34]

Investors in a project such as FLAG could capitalise their Indefeasible Rights of Use (IRU). An IRU does not mean that the holder owns and operates a set section of cable. What an IRU accomplishes is the right of the holder to access that part of the infrastructure to the prejudice of everyone else if they so choose. More important, IRUs can be sub-leased and/or rented for dis-

crete intervals much like a condominium holding in a real estate transaction.

Historically, the majority of IRU holders have been operator/investors, but FLAG proves that such a scenario is no longer exclusive to telecommunications organisations. Given the fact that privately financed satellite systems—a fantasy just fifteen years ago—are both feasible and profitable, there is no conceptual reason why fixed-line telecommunications infrastructure should be considered a ‘special case’, outside the scope of normal capital investment by the market.

It could be that financial institutions syndicate an underwriting to buy IRUs on new infrastructure or purchase the interest of current IRU holders. They could then lease repackaged sections to multiple tenants who could change according to the goals of the freeholder.

This has been common among telecommunications operators since the beginning of submarine cables. A New York City based bandwidth broker explains the historical process: “I want to run a piece of fibre between here and Europe. It is going to cost me US\$200 million and nobody can float it internally. So the capital market produces a bunch of money and everybody buys an option on that line to finance it...now in order to make that pay, they have to fill that cable but in order to fill it now they go out to the reseller market who commit to, lets say, five million minutes at a particular rate. They in turn have their own customer base. Everybody has a tariff they’re bidding on and so that capacity is resold long before the cable is even lit.”[35]

A purchaser of a lease or option contract based around an IRU could be a firm or group of firms who wish to fix their communications costs in advance. As telecomms embeds itself further into business processes, the idea of buying telecomms services to support specific business functions or even individual projects has become increasingly imperative. Thus, a likely market could include service providers offering delay sensitive information such as video on demand or video conferencing applications to electronic market participants. These providers may see a need for reducing their exposure to price and access volatility but only as their markets require. Otherwise, infrastructure is no more than a sunk cost.

Contracts could be sold on the basis of connect time and volume (i.e. you have two hours access to 45 Mbps with a half hour’s notice); or regular points in time (i.e. you have 10 Mbps access from GMT14:00-16:00 MWF); or specific points in time (i.e. what will you pay for a 2 Mbs line on Mother’s Day?) And multiple permutations could follow. Once a revenue stream is secured based on an IRU, the contract holder could denominate and issue digital currency which is redeemable in perhaps a more high powered currency or

even promise a straight commodity swap.

A speculator in such a market could be a telecommunications operator, a financial house or a straight forward gambler who perceives that increased complexity in transaction dialectics between buyers and sellers offers substantial scope for betting on price changes for communications. Considering that 2Mbps submarine cable links between the UK and Spain have roughly doubled in price in the last two years, such a bet is not that far fetched.

Rapid price swings involving bandwidth are likely as telecommunications costs for transacting in electronic markets are as yet unknown because the majority of present network charging formulas are connection-oriented. Each call has a set-up phase during which a connection is established and maintained for the length of the call.[36] Conceptually speaking, this model assumes that no one else can use the circuit. Thus, only a single accounting record is needed regardless of the session’s duration.

It is also useless for an electronic market. Interaction through packet switched networks is connectionless. In a packet switched environment, the communications session is broken into discrete packets which traverse the network separately.[37] Accounting for server usage on the WWW requires a separate record for every ‘hit’ which adds up rapidly even if a user perceives a continuous session. If telephone style accounting were used, the equivalent of a one minute call could generate over 2,000 accounting records and a ten minute call could entail accounting for over 20,000.[38]

Another aspect of Internet-style traffic is the nature of the medium itself. The origin and destination of a message are simply fields on the header of a data packet. Such a packet can travel through multiple switches, networks or countries—sending and receiving—all during the same communications session. This means that a telecomms carrier is virtually unable to estimate the termination point for a stream of packets travelling to a trunk.

The upshot is that capacity planning on telecommunications systems has become an even more esoteric art than before. The former method of interrogating voice switches to build a model based on historical data does not work for Internet traffic. Aside from a lack of historical data, the statistics carriers can gather from switches only show what was the previous node and where is the next node a packet is to travel.

Thus, it is dubious that current practice employed to account for telephony will transfer to an electronic market. Currently, the majority of Internet access pricing is based upon a fixed monthly subscription for a given bandwidth. This has been due not only to govern-

ment subsidy but also to the nature of traffic. For much of the Internet's history, traffic consisted of mainly text media such as email or simple file transfer which was not delay sensitive nor graphically complex. It rode the peaks and troughs of the PSTN at minimal cost.

However, the explosion of network usage by more graphically complex applications has put serious pressure on public and private backbones. Such bandwidth intensive applications would create price volatility if the network were charged according to usage as opposed to the present system. Although usage based algorithms are being discussed along with traffic differentiation, there is no clear consensus as to the suitability of either for supporting ubiquitous electronic markets interacting across regulatory jurisdictions and involving operators who may or may not have established interconnection and accounting rate regimes.

This happening while the most important question facing electronic markets has yet to be resolved—how network charges are to be allocated in a transaction setting. If a customer calls up the network and accesses the site of a seller, then the customer is paying for the communications. If that customer then decides to make a purchase of a product or service, should the final price of the item include their communications costs? Would there be a discount for a customer who made a quick decision and thereby conserved resources or an extra charge for dithering over a purchase? How would network congestion in downloading the requested item be treated, as part of the overall price or would there be scope for express delivery? Who pays the communications costs for returning an information based item? Is there scope for a vendor to extend extra communications capacity as part of a loyalty program?

Thus, a transaction for even a simple item involves a complex communications dialogue that alternates between secure and insecure, priority and non-priority, flat rate or usage sensitive and so on. Therefore, the variety with which network resources are invoked cannot help but be a considerable influence over the eventual cost. Yet this communications scenario can become an asset in its own right.

While IRUs are infrastructure centred assets, another possibility for capitalising the network would involve traffic itself. Careful analysis of usage patterns by an investor could allow them to purchase discounted revenue streams. A currency issuer could approach a telecommunications operator or an Internet access provider and, for example, propose to buy 30% of that provider's next six months revenue stream for a discount today.[39]

The network provider has just received a significant tranche of revenue without a single co-ordination cost incurred. On the part of the currency issuer, they

now have access to a revenue stream which can be securitised and sold on the secondary market for a possible profit. However, they could also use that commodity bundle to back their digital currency.

The way this could work would be to buy the revenue stream at a discount and hedge the nominal amount with a futures contract. The provider then issues currency for the nominal amount, thereby enjoying seigniorage. On the other side of the futures contract would be a speculator betting that actual usage of the infrastructure will be higher than predicted. The risk and the reward to either would be found in the accuracy of their predictions.

By bidding on the physical infrastructure or revenue streams, currency issuers and speculators would unlock information about the real communications costs for an electronic market. Such price information would be valuable for guiding decisions as to where to allocate or build network resources.

A financial service provider with undisputed access to network resources is not going to be an operator of a telecommunications network, just as an investment bank trading petroleum futures does not own filling stations. Instead it is an investor who is assuming risk in order to control a resource underlying an economic paradigm. In that sense, management of a bandwidth based contract need not be different than management of any other securities contract.

### **Tomorrow's money?**

So far, this paper has dealt mainly with theoretical considerations. Thus, it would seem appropriate to conclude on a note of fact. Digital currency has been used since March 1996 on a commercial basis in Finland. EUnet, one of Europe's leading Internet service providers and Dicash have launched a system which allows consumers to make and receive real time payments over the Internet.

The new system allows users to visit a 'virtual ATM' on the World Wide Web and withdraw money directly from their bank account to their ecash 'purse'. They can then make electronic payments to each other as well to on-line merchants. If the launch succeeds in Finland, EUnet, working with major banks, intends to roll out the service during 1996, in more of the 41 countries where it operates.[40]

Let us imagine that the Finnish project succeeds and is introduced to other countries. Let us assume that the Dicash system is secure. We now have a situation where on-line merchants are configured to accept ecash payments coming from various countries where the electronic cash is backed by different national currencies. The merchant's software will recognise ecash as

ecash but is all ecash the same value? Will there be an exchange rate between ecash coming from Italy and ecash coming from Germany for a vendor in Finland? Or should the value of ecash be tied to the long-suffering Euro?

Go further and imagine that the software can handle differing exchange rates for ecash or that it is tied to a supranational currency. If ecash is anonymous and is being used freely across borders, what will be the new monetary policy?

No currency issuer—bank or not—has stated unequivocally whether their product is a commodity money (i.e. a commercial commodity at the same time); a credit money (i.e. it constitutes a claim against a physical or legal person); or a fiat money (i.e. it is the same as legal tender). Furthermore, there has been little debate on whether digital currency will pay interest, is subject to reserve controls, is legally binding for a contract, can be used to pay wages, is a valid for credit negotiation and so forth.

This suggests that the overall ‘success’ of digital currency will be tangential to its security. Thus, the prime challenge will be to evolve systems where the transfer of ‘value’ in a public electronic market is governed by many of the ‘money-like’ properties found in today’s currencies.

But this does not mean that the today’s currencies automatically qualify as tomorrow’s ‘money’. As the process of exchange adapts to the realities of electronic commerce, so will the value of digital currency. Therefore, the desirability an issuer’s asset base to instil confidence in an unconventional economy is the paramount concern—not the protocol through which tokens are exchanged. This implies that the objective value of money—any money—is subject to the process of continual renegotiation.[41]

While basing currency on IRUs or traffic streams are just scenarios, the important point to remember is that transplanting monetary systems to new environments cannot be isolated in technology and comes with a degree of risk. Those who will prosper from offering financial services will be those for whom electronic commerce offers a unique mode for business. They will structure their wealth and organisation to meet the unique needs of an electronic market, instead of rushing towards a digital answer to a traditional cost problem.

If one assumes that technology drives history—and there are persuasive arguments on either side—then one should be able to predict the risk and reward inherent in a new possibility simply by extrapolating from past experience. Yet historical analysis supports the view that change often takes place in quite information-poor and uncertain environments. The paucity of information available to decision makers powerfully con-

strains their ability to assess the consequences of technical advance. The reason is simple: acquiring new information is costly.[42]

Electronic commerce is about to pass from infancy to a sort of childhood. As the technical challenges of secure transmission of information—representing value or not—begin to show a defining trend, the more basic questions of what underpins that information’s attractiveness in an electronic market setting become critical.

This implies a new conceptualisation of both the role of telecommunications and financial intermediation in a public electronic market. Those who understand how to leverage the communications resource required by all participants in an electronic market will be well placed to survive and flourish the coming shakeout of industry structure—telecomms and finance—that the new medium has all but assured.

## Notes

1 Office of Technology Assessment, “Electronic Enterprises: Looking to the Future” (Washington D.C.: U.S. Government Printing Office, OTA-TCT-600, May 1994).

2 Information Infrastructure Technology and Applications (IITA) Task Group, National Co-ordination Office for High Performance Computing and Communications, February 1994, pp. 13-14.

3 T. Malone, J. Yates, and R.Benjamin, “Electronic Markets and Hierarchies,” CACM 1987, p. 485; and A. Picot and C. Kirchner, “Transaction Cost Analysis of Structural Changes in the Distribution System: Reflections on Institutional Developments in the Federal Republic of Germany,” Journal of Institutional and Theoretical Economics 143 (1987): pp. 62-81.

4 O. Williamson, “The Modern Corporation: Origin, Evolution Attributes,” Journal of Economic Literature XIX (1981): pp. 1537-1568.

5 Malone et al 1987.

6 J. Yannis Bakos, “A Strategic Analysis of Electronic Marketplaces” in MIS Quarterly, Vol. 15, No. 3, September 1991, p.295.

7 Bakos, p. 297.

8 Bakos, p.297.

9 R. Benjamin and R. Wigand, “Electronic Markets and Virtual Value Chains on the Information Superhighway,” Sloan Management Review Winter 1995, p.63.

10 “Now it’s America Offline”, B&T Online 9 August, 1996, [<http://www.cromwellmedia.co.uk>]

11 “America Offline caused by Routing Error”, B&T Online 12 August 1996.

11 "America Offline caused by Routing Error", B&T Online 12 August 1996.  
[\[http://www.cromwellmedia.co.uk\]](http://www.cromwellmedia.co.uk)

12 Stephen H. Haeckel and Richard L. Nolan, "The Role of Technology in an Information Age: Transforming Symbols into Action", in The Knowledge Economy: The Nature of Information in the 21st Century, (Institute for Information Studies, 1993), p. 2.

13 Haeckel and Nolan. p.8.

14 Daniel Bell, *The Coming of Post Industrial Society*, (New York: Basic Books), 1973.

15 Georg Simmel, *The Philosophy of Money*, revised edition. trans. Tom Bottomore and David Frisby. (London: Routledge, 1990 [1978. Revised German Edition 1907]), p. 54.

16 Simmel, p.179.

17 Simmel, p.180.

18 Ludwig von Mises, *The Theory of Money and Credit*, Trans. H.E. Batson. (Indianapolis: Liberty Classics, 1980 [1934; 1st German edition 1912]), p.85.

19 John Kenneth Galbraith, *Money, Whence it came, where it went*, Second Edition (London: Penguin Books, 1995), p. 63.

20 E. Fama, 'Banking in the theory of finance', Journal of Monetary Economics 6(1) 1980, pp. 39-57.

21 Von Mises, p.293

22 L. Jean Camp, Marvin Sirbu, J.D. Tygar, "Privacy, Anonymity and Reliability in Electronic Currency Transactions", Department of Computer Science Working Paper, Carnegie Mellon University, January 1, 1995.

23 Camp et al, p.2.

24 Camp et al, p.2.

25 Elinor Harris Solomon, "Today's Money: Image and Reality", *Electronic Money Flows: The Molding of a New Financial Order*, Elinor Harris Solomon ed. (Boston: Kluwer Academic Publishers, 1991), pp. 29-30.

26 Solomon, pp. 29-30.

27 Paul B. Henderson Jr., "Modern Money," in *Electronic Funds Transfers and Payments: The Public Policy Issues*, edited by Elinor Harris Solomon, (Boston: Kluwer Nijhoff Publishing, 1987), pp.17-18.

28 Henderson, p.21.

29 Solomon, p. 39.

30 Solomon, p. 37.

31 Solomon, p. 37.

32 John du Pré Gauntt, "Undersea fibre: from toll road to trading pit", Public Network Europe, July/August 1995, p. 31.

33 Gauntt, p. 31.

34 Gauntt, p. 31.

35 John du Pré Gauntt, "Selling bandwidth by the pound", d.comm personal information server, The Economist Group, August 1995, (<http://www.d-comm.com>)

36 Hal Varian and Jeffery K. Mackie-Mason "Some FAQs about Usage-Based Pricing"(<ftp://gopher.econ.lsa.umich.edu/pub/Papers/usFAQs.html>)

37 Varian and Mackie-Mason

38 Varian and Mackie-Mason.

39 Derek Nicholas, Nicholas Associates, Interview with author 15 July 1995.

40 EUnet Press Release, "Europeans Can Now Make Cash Purchases On The Information Superhighway", 13 March 1996, <http://www.eu.net>

41 Bloch and Parry, *Money and the Morality of Exchange*, Cambridge University Press: Cambridge, 1989, Introduction p.23.

42 Nathan Rosenberg, *Exploring the black box: Technology, economics, and history*, (Cambridge:Cambridge University Press, 1994), p.5.

# GENERIC ELECTRONIC PAYMENT SERVICES

## *Framework and Functional Specification*

Alireza Bahreman

*Principal Investigator, E-CO System Project<sup>w</sup>*

*bahreman@eit.com*

### **ABSTRACT**

*We present a framework for integrating electronic payments with applications. We call it the Generic Electronic Payment Services (GEPS). There are five payment services identified in our framework. The model provides maximum transparency to the application developer, while maintaining maximum consistency for the end-user. This is achieved by a layered model where the application (top layer) is isolated from the details of the payment services (lower layer). We also believe that this framework makes it possible for new and innovative payment solutions to more easily integrate with existing applications, resulting in increased competition without stifling innovation. This ultimately leads to higher quality and cost effective solutions which the market will choose.*

## **1 Introduction**

The excitement surrounding Electronic Commerce has generated a lot of creativity. Numerous payment mechanisms and system providers are formed. Some have even identified new paradigms for payments in an online world. Many payment protocols have been proposed, the most notorious of which is the Secure Electronic Transaction [SET] protocol proposed by the credit card associations. There are numerous other systems<sup>1</sup> proposed. However, it is beyond the scope of this paper to enumerate a comprehensive list of these systems. There are simply too many choices and this is likely to cause problems for both application developers and end-users alike who wish to use electronic payments.

As a direct result of this creativity, application developers are finding it harder and harder to integrate with and use payment systems. End-users are also confused with the abundance of choices and the differences in the approach taken by each solution. There is no consistency between the user interfaces of the various payment systems making it harder for the end-users to understand and use. Due to the difficulties, application developers are reluctant to integrate with new payment systems and protocols. Therefore, an

inventor of a new system faces steep barriers to entering the market. Once one or two dominant systems are integrated, others are blocked out of the market. This results in stifling innovation and monopoly of a few solutions which ultimately leads to inefficiencies for the end-users at higher cost.

What is needed is a comprehensive and consistent framework to capture the essence of electronic payments. Complying with this framework solves the programmer/end-user/creator problems mentioned above. It allows payment services to be abstracted from the application programmer providing him/her with a simple Application Programming Interface (API) resulting in source-level portability of code across different payment systems. It provides end-users with a consistent look-and-feel. It also provides creators with the hooks needed to plug their inventions into existing applications, therefore removing the barriers to entry. This leads to benefits in the long-run for the end-users.

In this paper, we propose such a framework. We take a top-down approach to solving the problem. Using a layered model, we identify applications, infrastructure elements, services, and resources in the framework. We then specify the main functionality of the individual

<sup>w</sup> This work has been funded by DARPA contract DABT63-95-C-0133.

<sup>1</sup> For a list of pointers to these works, please see <http://eco.eit.com/information>.

services. It is important to note that a full implementation of this framework is needed before one can be assured of its completeness. We have started a partial implementation of the framework in the E-CO System project [WebSite] and have documented the specifications for it in a companion paper [PayNegot]. In joint work with the Stanford Digital Library Initiative, we have further specified [U-PAI] parts of the GEPS framework by defining a mechanism-independent API to insulate the application programmer from the payment-specific details.

We discuss the framework in greater detail in Section 3. There, we also consider the general phases in which an application would conduct electronic payments. Keeping these phases in mind would help confirm the completeness of our functionality specifications in later sections. In particular, Section 4 presents the GEPS focusing on the functionality and operations and not necessarily the details of the API parameters and arguments. Those details will be provided in a future document on our website under the Project Information section. We discuss our GEPS implementation and provide a usage example in Section 5. In Section 6, we present a brief description of related work and compare the approaches. We conclude with closing remarks in Section 7, followed by acknowledgments and references. In the next section, we present the terminology used in this paper.

## 2 Terminology

In this section, we present definitions for some of the terminology we use consistently throughout this paper. This is simply intended to provide an aid to understanding the discussions in this paper.

### 2.1 Payment Method

Payment method is a high-level categorization of a payment transaction. A way of making a payment, if you will. Some known payment methods include: Cash, Credit, Debit, Electronic Check, and Electronic Cash. For example, Alice might pay Bob ten US dollars in paper currency; she is therefore using the Cash payment method.

### 2.2 Payment Protocol

Payment protocol refers to a collection of messages used to carry electronic payment related information and instructions among the parties involved and the flow/sequence of those messages. As an example of an electronic payment protocol, consider the SET specification.

### 2.3 Payment Mechanism or System

Payment mechanism or system refers to a complete system designed to enable and execute payment transactions among parties. Some payment mechanisms require a third party intermediary such as [NetBill] and [NetCheque].

### 2.4 Payment System Provider

Payment system provider refers to the party operating a payment system or mechanism. For example, the company CyberCash, Inc. is a payment system provider, for their credit card payment system [CyberCash].

### 2.5 Payment Capability

Payment system providers often implement the software needed to use their payment mechanism. This software is then used by other developers in their application to conduct electronic payments. These implementations are referred to as payment capabilities. The nature of their implementation can vary (e.g. from being an applet to a shared library or even a plug-in). The end result, however, remains the same—enabling an application to use services provided by a payment system.

### 2.6 Payment Service

Payment service refers to a collection of payment functionality grouped in, or implemented as, a module used during a payment transaction by applications. There are five major payment services identified in this paper. They are Payment Interface Management, Payment Method Negotiation, Preference Management, Capability Management, and Transaction Management. We discuss each in length in Section 4.

### 2.7 Account and Account Proxy

An account is where value is gathered. Alternatively an account may provide its owner with the privilege to use a payment system. In the physical world, we are very familiar to the concept of an account (e.g. a bank account). In the online world, there may be new types of accounts all with similar purposes as in the physical world. An account proxy, however, is the online representation of a physical world account. The information in the account can be cached in the account proxy (e.g. the running balance). However, the account proxy's data is not necessarily up to date. One can close or destroy an online account, and an account proxy. However, physical accounts cannot be destroyed in a pure online fashion and would require out-of-band mechanisms.

## **2.8 Payment Instrument**

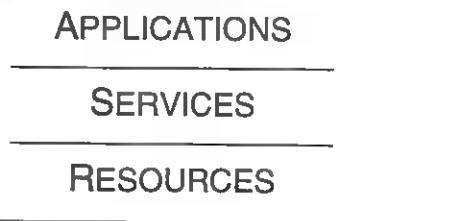
A device or token which is used during the payment transaction. For example, a particular Visa card with an account number and an expiration date is a payment instrument used in a transaction of type Credit payment method.

## **■ Framework**

In defining our framework, we take a top-down approach. Rather than starting at the API level, we first consider the applications and the infrastructure required to support electronic payments. We then identify the main payment related services that would be used by those applications. We enumerate the functional requirements of each payment service. We also identify the resources used by each service in order to provide the service functionality.

This leads to a layered model consisting of a separate layer for Applications, Services, and Resources. The diagrammatic representation of this layered model is provided in Figure 1. The Applications layer consists of any software program that utilizes payment services. This includes general applications, such as web browsers, as well as highly specialized payment related applications. For example the software used in a bank with which electronic coins are minted. The Services layer encompasses all the payment specific services and functionality. The Resources layer provides the building blocks needed to implement those services.

**Figure 1- The Layered Model**



In this model, an application is isolated from the specific details of each of the payment services. The Services layer provides the abstraction necessary for the application programmers to use payment services without knowing the implementation details. Due to this layer of abstraction, an application need not change dramatically to take advantage of a variety of payment mechanisms. The payment services, in turn, manage a set of resources in order to implement their functionality. The resources are created, manipulated, and destroyed by the services. Applications do not directly access the resources.

We discuss each of the three layers in more detail in the following sections. We also discuss several phases of operation for a typical application which uses electronic payments.

## **3.1 Applications Layer**

The Applications layer is the means for abstracting all applications and programs that use the payment related services. This includes general applications that only use payment services, and specialized applications that provide the infrastructure required to support the payment process for all other applications. An example of a general application is a simple online catalog or a storefront. This application is mainly concerned with presenting the end-user with the inventory of the goods provided by the store. The payment phase only occurs if and when the end-user decides to make a purchase. If there was no electronic payment service available, the store would probably direct the end-user to the phone number for a customer service agent who could help the end-user. If, on the other hand, an electronic payment service was available, the program would utilize it to make the end-user pay for the goods. Once the payment is complete, the application could provide for the delivery of the goods independent of the payment phase.

The specialized applications (also referred to as infrastructure applications) are necessary to support electronic payments for all applications. These applications are also implemented using the same payment services that are made available to other applications. The difference is that the functions they perform are in support of the payment services used by all applications. For example, a Bank software is one such infrastructure application. The function of the Bank software is to provide access to financial accounts maintained by itself. Also, a Bank stores and in some cases creates value (represented by electronic money). All these functions are necessary for all applications which use electronic money, therefore, the Bank is considered to be an infrastructure application.

There are several categories of infrastructure applications that must be considered. They are:

- Trusted Third Parties
- Banks
- Brokerages
- Traders
- Government

Trusted Third Parties (TTP) are a class of applications that act as intermediaries during a payment transaction.

The specific function of the TTP depends on its application domain and purpose. Some known TTPs include GCTech's [GlobeOnline], NetCheque's payment server, and NetBill's server.

Banks, Brokerages, and Traders could also be considered as TTPs. The only difference is that their functionality is very specific to electronic payments and can be used at any time during or after a payment transaction; in other words they are not necessarily mediating each and every transaction. The functions performed by Banks, in the physical world, are well known and are very similar to those in the online world. Banks hold value and in some instances create value. In our model, Brokerages are TTPs that are used for micropayments. Brokerages act similarly to Banks by creating micropayment tokens which in a way represent value. Traders here represent TTPs which can convert value from one format to another. These are functions similar to the currency exchange bureaus we are all familiar with in the physical world.

The infrastructure applications that represent the Government include the Internal Revenue Service and any other governmental function that must be intimately tied to the payments infrastructure. Even the law enforcement agencies might require functionality that must be supported by the infrastructure (for example the hotly contested Key Escrow service may be required for financial institutions' keying material).

We must note that there are other infrastructure applications which are necessary for electronic payments. For example, Certification Authorities are needed for cryptographic keying material used in the payment transaction. These set of infrastructure applications fall in the realm of general security and are beyond the scope of this paper. However, at E-CO System project we are working closely with Trusted Information Systems on defining and implementing such applications as well [PKI-API].

Regardless of the details pertaining to any specific application that falls into one of the above categories, the fact remains that these infrastructure applications are essential for electronic payments. Therefore, the framework must accommodate them. Figure 2 illustrates an instance of the earlier layered model and is revised to include the specific payment related applications, services, and resources.

Ideally, our model will accommodate all types of general applications. The model works especially well with applications in which payment functionality is a separate and modular part of the processing. One can

distinguish the payment functions from the rest of the application's tasks. There could be several payment related functions in an application, but they would all be independent from the application logic. For example, a simple and common application of payments is the purchasing of goods. An application might direct the end-user through a shopping phase in which the end-user decides on the goods to purchase. Once this is done, the application begins the distinct functions of payment. Once the end-user pays, the application returns to the shopping process by displaying a payment confirmation for the end-user. Therefore the payment operations are completely detached from the rest of the application.

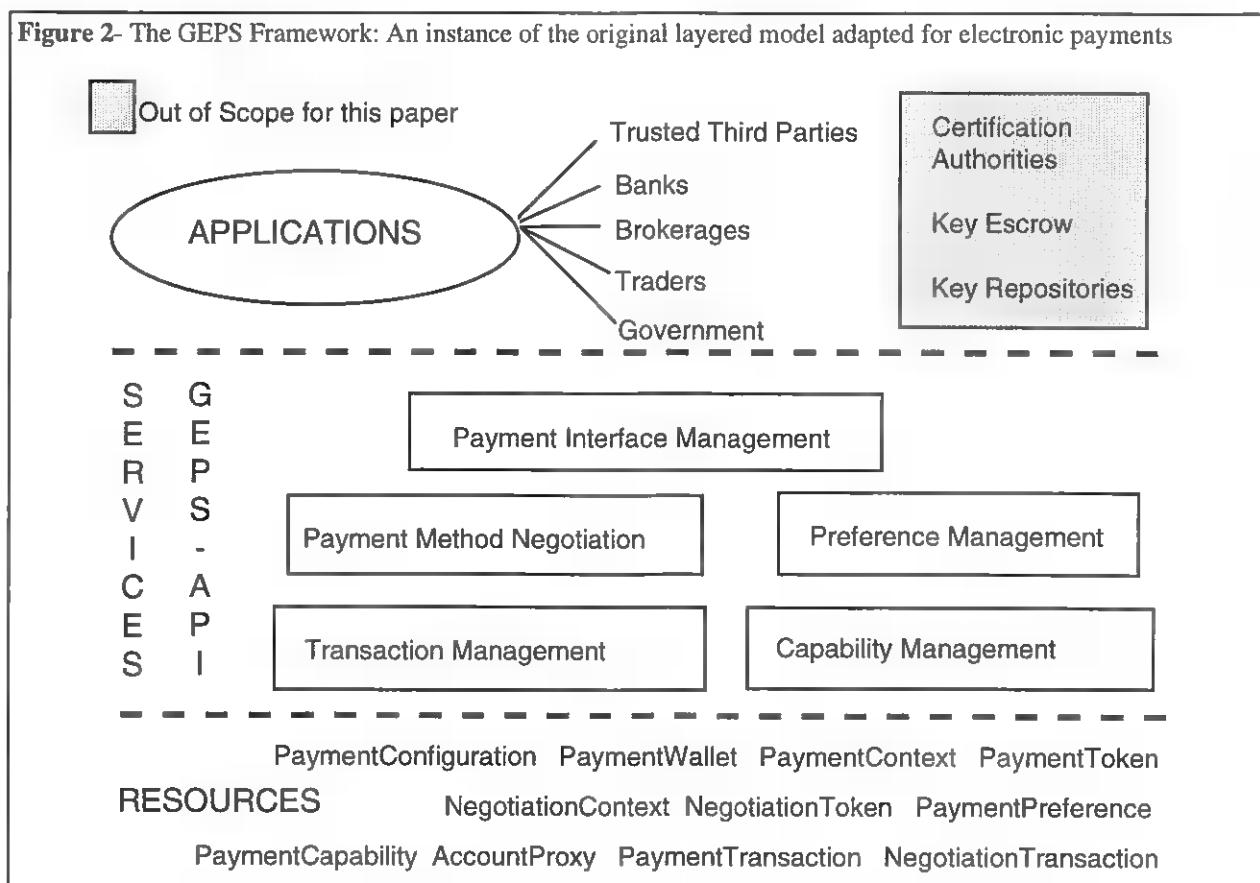
There are, however, sophisticated applications in which the payment processing is not completely isolated (and cannot be separated) from the rest of the application. For example, the NetBill trusted intermediary combines the delivery phase with payment phase for shopping in order to provide a mechanism referred to as certified delivery. (Certified delivery ensures that the delivery of the goods and payment are completed atomically.) While we have not yet analyzed the impact of our framework on such applications in depth, we believe that it would apply equally useful for such applications. If necessary, these applications could be provided with a more granular interface for accessing the Resources layer. For example the Payment Interface Management could be bypassed and direct access to the Transaction Management would be utilized instead (this example will become clearer once we discuss the services in the next section).

### 3.2 Services Layer

In order to expand the discussion on the model presented in Figure 1, we move on to the next layer—Services. We have identified several payment related functions and have categorized them in a modular group of five services: Transaction Management, Capability Management, Preference Management, Payment Method Negotiation, and Payment Interface Management. We call them the generic electronic payment services, payment services or GEPS for short.

Applications differ on how much they rely on or use any one or more of the GEPS. In order to have a comprehensive application programming interface, however, one has to take all of these services into consideration. The functionality of each and every one of these services are explained in greater detail in Section 4. Here, we provide a brief summary to help clarify Figure 2 which pictorially illustrates the relationship between these services and the other layers.

**Figure 2- The GEPS Framework: An instance of the original layered model adapted for electronic payments**



Transaction Management provides logging, audit trails, receipts or records of transactions, and the status of a transaction. Ensuring transaction atomicity, as well as recovery, are also provided.

Capability Management is the link GEPS and other payment system providers. It provides the interface to install and remove payment capabilities of different payment systems as well as ensuring smooth transition to and from those payment systems.

Another critical component of the system is the Payment Method Negotiation. This is increasingly necessary in the light of the variety of payment mechanisms becoming available. It manages the mechanism for transacting peers to negotiate and select the method with which they want to pay one another.

Preference Management is where the end-user's preferences are maintained. Some of these preferences could be regarding his/her spending limits, or choice of payment system or method. These preferences are registered and used in the payment selection process by the Payment Method Negotiation module. In addition, the transaction management module could use some of these preferences to accept or reject a transaction.

The Payment Interface Management is a layer of abstraction with the primary goal of making the use of payment services even simpler for the general application. More sophisticated applications can bypass this layer for the most part and directly access the underlying services. The Payment Interface Management has access to, and is built over, all other payment services.

We believe that this set of services represent a complete set and would provide all the functionality required by applications to conduct electronic payments. However, depending on where a line is drawn between the application logic and the payment logic, one might find a functionality that is not covered by these services. For example, one might consider a shopping model in which the price negotiation is tightly coupled with the payment transaction. In that case, the services identified above would have to be extended to accommodate the price negotiation for the shopping model. Needless to say, we have considered a great deal of functionality and feel that the above services cover most, if not all, of the required functionality.

### **3.3 Resources Layer**

The payment services provide their functionality using a set of resources. Only the GEPS have direct access to these resources. Applications can only use the services to perform operations on the resources. Table 1 presents a comprehensive set of resources used by each service. For additional information on each resource, please refer to the Appendix where the primary functions of each resource are enumerated. Some of these resources have been displayed in Figure 2 above.

### **3.4 Phases of Operation**

Before discussing the functional specification of the payment services, it is important to realize that a typical application using electronic payments might include several phases. This is meant to be a complete set. Any application using electronic payments will function in one or more of these phases at any time. All operations and services discussed in the next section must support these phases (when applicable) for completeness. These phases are enumerated below:

- Registration and Initialization
- Inquiry
- Payment (positive usage)
- Dispute (negative usage)
- Abort or Cancel
- Trace and Audit
- Recovery
- Closure and Destruction

The first phase of all application systems requiring payments is the Registration and Initialization phase. Registration refers to the process of setting up an end-user to access financial data. Opening a bank account, acquiring credentials for authentic payment processing, and registering with a payment system provider are all examples of processes conducted during Registration. Registration often involves communicating with another party in a distributed network. During Initialization, on the other hand, an application is typically performing local operations. For example, it may be setting up configuration files or creating records.

**Table 1- The Service/Resource Relationship**

<b>Services</b>	<b>Resources</b>
PIM	PaymentConfiguration, PaymentWallet, PaymentInstruction, PaymentDispute, PaymentContext, PaymentToken, ErrorTable
MN	IOProcessor, UserInterface, NegotiationContext, NegotiationToken
PM	PaymentPreference, PreferenceCriteria
CM	PaymentCapability, Account, AccountProxy, ValueToken, CapabilityContext, CapabilityToken
TM	PaymentTransaction, ReceiptTransaction, DisputeTransaction, NegotiationTransaction, PreferenceTransaction, CapabilityTransaction

The Registration and Initialization take place once per payment system or once per transaction, depending on what is being registered or initialized. Consider for example opening an account with a bank which accepts electronic money. It is likely that the registration occurs once in this example. Once the bank account is opened, the end-user may use this account for all payment transactions. Each time a transaction is initiated, there might be a transaction record that must be initialized. Clearly, this initialization operation is performed once per transaction.

The Inquiry phase is when the application is making financial related queries on behalf of the end-user. These queries include, but are not limited to, the status of a payment transaction, the information on an account (e.g. balance), and the list of payment capabilities of another party. This phase is most common among applications and can occur at any time before and/or after, and in combination with other phases.

Another common phase is Payment (or positive usage). Here the application allows the end-user to begin a payment transaction with another party. The reason we refer to this as a “positive” usage is to distinguish it from an opposite but similar phase—Dispute. During the Dispute (or negative usage) phase, the application assists the end-user in disputing an existing transaction and often returning items in exchange for a refund.

Payment transactions are atomic, meaning that they are performed only once or not at all. The atomicity is guaranteed by the transaction management module and through the programming interface. The application must commit to a transaction it starts in order to allow the transaction to be later recovered in case of temporary failures. Once a payment transaction is committed to, the application may not abort it. Before a transaction is committed, the application is always able to cancel/abort it. This is the Abort or Cancel phase. The application must always end a transaction to ensure proper closure.

The Recovery phase is when a payment transaction that has been prematurely interrupted is restarted and completed successfully. Transactions can fail due to a computing or communication failure. If the transaction was started but not committed to, the application cannot recover the transaction. However, if the transaction was committed to, but not ended, the application can recover the transaction using the logs. Recovery occurs after the computing or communication failure is resolved.

During Trace and Audit applications can perform a check on a past transaction (in order to ensure its accuracy) and generate audit trails and/or reports. The GEPS framework's logs are used for such operations. The application usually performs trace or audit after other phases have been completed.

The Closure and Destruction phase refers to the final phase of operation for applications. Closure refers to operations in which access to data, or data themselves, is temporarily removed from the application (much like closing a file). Examples are closing access to an account, closing the session to a peer, and closing the application itself and all its payment interfaces. More permanent closures are performed during the Destruction phase. This includes deleting a record of a transaction, permanently closing an account, and removing a preference or other configuration data from the GEPS environment files.

An understanding of these phases is essential and very helpful for designing the functional specification which follows. There, we make sure that all these phases are supported through the use of one or more of the GEPS.

## 4 Functional Specification

We now migrate from discussing the framework to specifying the functionality of all the individual Generic Electronic Payment Services (GEPS). There are a total of five services identified: Transaction Management (TM), Capability Management (CM), Preference Management (PM), Payment Method Negotiation (MN), and Payment Interface Management (PIM). Each service is described individually in greater length in the following subsections. We discuss the following for each of the services:

- **Functionality-** describe its main functionality.
- **Relationship-** which other services it uses and by which services it is used.
- **Phases-** during which of the above mentioned phases of operation it is used.

- **Connectivity-** communication requirements to its peers and to other application processes.
- **Security-** discuss its security requirements.

Note that the connectivity requirements have a substantial impact on the implementation of the GEPS. For example, consider the case where a GEPS peer-to-peer communication is required. Either the application using the service must provide for the communication, or the service itself. In order to achieve transport independence, the services in our model are not tied to a communication network. These services are therefore not capable of transmitting data between peers. This means that the application must transmit the data for the services. For every service, this is done in our model by maintaining a *context* for the state of the communication, and by generating *tokens* as messages between peers that need to be transported. The context maintenance for our services is similar to the notions presented by the Generic Security Services work [GSS-API]. Simply stated, a context or state is maintained by each peer and regularly updated as a result of peer-to-peer communication. The peers communicate by sending tokens to each other. These tokens are opaque to the application and can be sent to the peer through any transport mechanism. Applications must repeat sending tokens back and forth between peers until the service reports the completion. In our framework, peer-to-peer communication affects some, but not all, the services.

The current version of the GEPS Application Programming Interface (GEPS-API) classes have been provided in the Appendix. Tables (A) through (E) in the Appendix present the high-level object classes of each service and their main functionality.

### 4.1 Transaction Management (TM)

The Transaction Management component of the framework is responsible for the services related to maintaining transaction sessions. It ensures highly reliable operations (through transaction atomicity) and facilitates recovery of transactions that have failed due to network or computing failures. It provides the status of the transaction, and maps acknowledgments or receipts to their relevant transactions. It also provides the ability to generate audit trail and audit reports.

The transaction atomicity guarantees made by the TM cover the access and use of payment capabilities and do not extend to the actual payment systems' transactions. For example, consider a payment transfer from one bank account to another. The TM starts a transaction for this event when the payment capability initiates the

transfer. Assuming that the communication to the Bank succeeds, the Bank will start its own transaction to complete the transfer. If the Bank succeeds, it will send an acknowledgment back using the payment capability. If the communication network fails, however, the GEPS environment would not receive the acknowledgment and would continue to consider the transaction as not completed. The GEPS can then recover and complete the transaction by resubmitting the request to the bank. The Bank, of course, will verify its independently maintained transaction log and realize that the transaction has already been completed and resubmits the acknowledgment.

The TM is used in all phases of operation. All payment services use TM to record their events. The PIM provides a user interface for the functionality of the TM. The PIM includes a checkbook register where the end-users can see all transactions completed (or in process) at any given time. The end-users can easily dispute these transactions or query about their status. The transaction audit and reporting functionality is provided through the PIM.

All TM operations are local and do not require peer-to-peer communications. Since the entire system depends on the reliability and robustness of the TM, however, the security requirements of the TM are of critical importance. Notably, the integrity and authenticity of the log records must be maintained. In multi-user environments, this is achieved by signing and encrypting the log information and requiring end-users to login in order to modify or use the log.

#### **4.2 Capability Management (CM)**

The Capability Management is the interface between the GEPS and external payment systems. The CM ensures that the interface is consistent and provides a mechanism to transition from the application to the payment capability and vice versa. Recall that payment capability is the implementation of a payment system or mechanism.

In our framework, each payment system is represented as an instance of an object—the PaymentCapability.

The code for this object is either provided by the payment system provider or is provided by the GEPS implementation. Either way, this object performs the functions necessary to glue the payment system to the GEPS framework. When new payment systems register, a record is created and maintained on each PaymentCapability. These records are used by the CM to provide the MN with a list of potential payment capabilities. These records are represented in the system by variable length tuples of tag/value pairs. Each tuple includes tagged information such as the payment method, mechanism, protocol, and account as well as other relevant information. Each tuple uniquely identifies a PaymentCapability with all of its related payment instruments, payment accounts, etc. There can be more than one tuple for a particular payment mechanism. A payment mechanism may have several accounts resulting in multiple payment capability tuples. Multiple payment capability tuples may share a common account. Table 2 presents some valid payment capability tuples and their associated description. (Tuple tags have been omitted due to space constraints.) As an example, consider the tuple (method = credit, system = CyberCash, protocol = SET, provider = BoA, account=1234-34-9999). This tuple represents the PaymentCapability of a Credit payment method using CyberCash's implementation of the SET protocol with a BoA account. On the other hand, the tuple (method = credit, system = CyberCash, protocol = SET, provider = BoA, account = 5555-39-2222) represents the use of SET protocol with a different account at the same bank. Both tuples require the use of the same payment mechanism.

All payment capabilities installed in the environment register with the CM during the Registration and Initialization phase. The CM is also used when assisting the end-users obtain an account with a payment system. The other phases in which the CM is utilized include the Dispute, Payment, and Inquiry. The CM is used by the PIM and MN services. The CM uses the TM to record events including registering new payment capabilities and probing made by the MN regarding payment capabilities.

**Table 2- Sample PaymentCapability Tuples**

<b>PaymentCapability Tuple</b>	<b>Definition and Meaning</b>
(credit, VeriFone, SET, BoA, VISA, 1234-34-9999)	Visa credit card using VeriFone's SET impl.
(credit, CyberCash, SET, BoA, VISA, 1234-34-9999)	Visa credit card using CC's SET implementation.
(credit, CyberCash, SET, BoA, MC, 5555-39-2222)	MasterCard credit card using CC's SET impl.
(credit, FirstVirtual, FV, John Doe: doe@foo.com)	First Virtual mechanism on John Doe's account.
(cash, DigiCash, ecash, Mark Twain, 1234-3434)	Cash using DigiCash's ecash on MT acct.

All operations in CM are performed locally. The only non-local and peer-to-peer communication occurs within the boundaries of the PaymentCapability objects. The PaymentCapability must communicate to its peer or server on other machines to complete the financial transaction. The transaction could be related to making payments or disputes as well as making an inquiry (e.g. account balance). For these peer-to-peer communications, a context and token are maintained by each PaymentCapability. Some payment systems might prefer to manage their own peer-to-peer communication, in which case the token and possibly context might not be provided to the GEPS framework.

The security requirements are handled by the individual payment capability implementations and vary from system to system. The GEPS can ensure the integrity and authenticity of the code it runs for each of the PaymentCapabilities by running signed and authentic code. If the environment on which the GEPS is running is trusted, one only needs to ensure the authenticity of the PaymentCapability code once, at installation.

#### **4.3 Preference Management (PM)**

The end-users' preferences regarding payments are maintained by the Preference Management. These include the users' preferred choice of payment method (e.g. Cash vs. Credit), payment mechanisms and systems (e.g. First Virtual vs. CyberCash), and account or payment instrument (e.g. Visa vs. MasterCard). The PM provides the means to sort, in order of end-users' preference, the list of payment capabilities generated by the CM. This functionality facilitates the negotiation and selection of the payment capability to use in a transactions.

The end-users' preferences are stored in a series of records which are called PaymentPreferences. These records are either an ordered list of payment capability tuples, or they indicate the conditions under which a particular payment capability is desirable. Each preference either fully specifies a payment capability tuple or uses wildcard mechanisms to match multiple tuples. One or more PaymentPreferences records are grouped according to a criterion such as "total price". The group is maintained by what is called a PaymentCriteria. There can be several criteria in the system. Some of the criteria we have considered are:

- choices—payment method, protocol, etc.
- spending limits—a certain maximum daily value
- total price—value of a transaction
- transaction cost—usage cost of payment capability

This is best explained in an example. Table 3 depicts the case where the end-user has specified the following preferences: Under the criterion "payment method choice", the end-user has specified an ordered preference of micropayment, cash, and credit. This implies that the end-user prefers to use micropayment over cash over credit cards. As for the criterion "total price", the end-user has expressed that only charges below \$1 use micropayments, below \$10 to use cash, and over \$10 to use credit.

**Table 3- Sample PaymentPreferences/PaymentCriteria**

<b>PaymentCriteria</b>	<b>PaymentPreference(s)</b>
choice:method	{condition: ordered-list; (method = micropayment), (method = cash), (method = credit) }
total_price	{condition: \$value > 10; (method = credit) } {condition: 1 < \$value < 10; (method = cash) } {condition: \$value < 1; (method = micropayment) }

The PM uses the TM to record any changes made to the preferences, and any other operations which use the records which it maintains. The PM is used by the MN to sort the list of payment capabilities created by the CM during the payment method negotiation and selection. The PM is mainly used during phases in which the MN is used—Registration and Initialization, Payment, and Dispute.

All operations performed by the PM are local and require no peer-to-peer communication. The security requirements are integrity and perhaps confidentiality of preference information. Assuming that the local host is trusted, there is no security mechanism needed. Otherwise, normal signing and encrypting of the preference data, and authenticated access through login is sufficient.

#### **4.4 Payment Method Negotiation (MN)**

Payment Method Negotiation entails more than the negotiation of payment methods to use in a transaction by the communicating peers. Payment Method Negotiation is the process of negotiating and selecting the payment capabilities and determining the common payment methods, protocols, mechanisms, system providers, and accounts or account proxies. The

negotiation enables peers to determine what they share in common. The final step is the selection. During this step the payment method, protocol, mechanism, capability, system provider, and account or account proxy to use to make a payment is determined. The negotiation could occur just prior to making a payment or at any time as an inquiry into the peers' payment capabilities. The selection must occur before payment can be made.

The MN must balance the delicate tradeoff between the end-user's right to select vs. the complication caused by the display of numerous choices to the end-user. Furthermore, the end-user might not be aware of the subtle differences in their payment related choices. For example, the end-user might not know the different payment protocols, but s/he will know and can understand the different accounts which s/he owns. Our recommendation is to use the end-user's preferences to trim the list of common payment capabilities during payment negotiation. To the extent possible, it is desired to use end-user's input only during the final selection. The end-user must be allowed to override this behavior. For example, under certain circumstances, the end-user might allow the selection of a micropayment capability without being prompted for it.

The MN uses the PM, CM, and TM services. It uses the end-user's preferences maintained by the PM to make the process of selecting the common payment capability as automated as possible with minimum end-user involvement. The MN uses the CM to list available payment capabilities and query them to see if they are valid candidates in the transaction for which the negotiation is being performed. Like other payment services, it uses the TM to record the status of a negotiation.

During the negotiation process, the MN uses the PM to sort the payment capability list generated by the CM. This is done according to a set of criteria. The application using the MN will indicate what the sequence of those criteria will be. More than one criterion can be mentioned which will be processed in a serial fashion during the sorting process. To complete the selection process, the MN might require end-user's input. It would be necessary for the end-user to interact with the system during the selection process if a single payment capability cannot be negotiated automatically.

This process is best illustrated in the context of an example. Based on the PaymentPreferences and PaymentCriteria provided in Table 3 above, assume the MN tries to sort a list of payment capabilities including DigiCash's ecash, VeriFone's SET implementation, and

Mondex's stored value cash system. Also assume that the total price of the items being purchased is \$5.75. When the first criterion (choice:method) is applied, the partially sorted list would look like: DigiCash, Mondex, and VeriFone. With the second criterion (total\_price), the VeriFone credit card solution is discarded, leaving two competing systems. Had there been a transaction cost criterion stating that Mondex costs 5% of the total transaction value vs. 3% for DigiCash, then only one would remain—DigiCash. Still, there might be several capability tuples enabling DigiCash and Mondex.

The MN service is primarily used during the Inquiry and Payment phases. If the negotiation occurs during the Inquiry phase, the results may not be used immediately. The GEPS provides context maintenance in order to retain the results of the negotiation for later use. On the other hand, if negotiation occurs during the Payment phase, the results will likely be used immediately. In either case, since peer-to-peer communication is required, each peer must maintain the communication context. Communication is performed using tokens which indicate the parties' intentions during each exchange.

The security requirements are integrity, authenticity, and possibly confidentiality. No non-repudiation is required. If no security is provided, an intruder could modify the negotiation messages resulting in the peers' failure to negotiate. This is a form of denial of service which is a known hard problem and cannot be fully avoided even if security measures were performed. For this reason, the use of security for the MN is left to the discretion of the end-user or application developer.

We have implemented a web-based Payment Method Negotiation service which allows web browsers and servers to negotiate common payment capabilities on behalf of the end-user. The details of our approach to MN is described in a companion paper [PayNegot].

#### 4.5 Payment Interface Management (PIM)

The Payment Interface Management provides the user interface elements of the GEPS allowing maximum abstraction for the end-user and the programmer. For example, the PIM provides a unified inquiry mechanism for account information and payment capabilities, end-user's preferences, peers' payment mechanisms, and transaction status. The PIM uses the CM, PM, MN, and TM services to accomplish its functions. See the individual services for more details including their connectivity and security requirements. Applications could bypass PIM and directly access interfaces of the individual services instead.

Due to its critical importance, the PIM is involved in all phases of operation. The following functionality is performed by the PIM for applications. For each functionality, we will specify the operational phases in which it is used.

- Configuration
- Wallet
- Value Transfer (positive and negative)
- Error Handling

#### 4.5.1 Configuration

The Configuration functionality entails the maintenance of the GEPS environment. This includes all default values for services, information on infrastructure applications, as well as the value of other parameters used by individual services. Most of the information is provided during setup by the administrator or during the registration of new payment capabilities. The end-user may also provide parts of the data. The main resource implementing the Configuration functionality is called PaymentConfiguration (configuration, for short).

The Configuration functionality is used during all phases of operation. All services use the configuration data to access environmental information. The configuration data is initialized or updated in the Registration or Initialization phases. During other phases, the configuration data is accessed mainly in a read-only fashion.

All operations performed on the configuration are local and do not need peer-to-peer communications. The security requirements are integrity, confidentiality, and in cases where the machine is timeshared among multiple end-users, authentication. No non-repudiation service is necessary. The integrity and confidentiality of the configuration data can be ensured by signing and encrypting the data with the end-user's keys. The end-user must login in order to use the information. The login period can vary and depends on the application.

#### 4.5.2 Wallet

The Wallet functionality facilitates the binding of accounts and payment mechanisms (represented as payment capability tuples), with associated user-related information (e.g. name and address). The primary resource providing the Wallet functionality is an object called PaymentWallet (wallet, for short). Using the wallet more than one payment capability tuples can be grouped together providing the ability to create and maintain multiple end-user profiles (e.g. personal and corporate).

The wallet is primarily used during the Payment and Dispute phases to provide the value to transfer or receive. During the Registration and Initialization phases, the wallet will be set up and new payment capabilities and accounts will be added to it.

The wallet provides an interface for the end-user to save a record of all payment transactions, initiate disputes, and make status inquiries. During the Inquiry phase, the wallet provides consolidated information on the accounts and payment capabilities for one or more end-users. The wallet also provides an interface to the end-user with which to terminate a transaction in the Abort or Cancel phase. During the Closure and Destruction phase the application could either temporarily close or permanently destroy the wallet.

The wallet also provides an interface to recover transactions using the TM. During the Recovery phase, the wallet component of the PIM uses the TM service to implement this functionality. There are numerous reasons why a transaction would fail to complete. Mainly, the communication and networking problems may result in loss of messages. Any computer failure may also lead to incomplete transactions. The goal of the Recovery functionality is to recover incomplete transactions once the computing or communication problems have been resolved. Only transactions that were committed (but not ended) would be candidates for recovery. Incomplete transactions would be lost. Note also that not all transactions are suitable for commit operations. For those transactions, the call to the commit operation will result in a warning indicating that the operation cannot be committed and may be lost in the event of a failure. The recovery process is mainly a repeat of the operations that were committed but not ended. Therefore, the security or connectivity concerns are the same as those for the operations themselves.

In support of the Trace and Audit phase, the Wallet functionality provides the ability to generate audit trails and reports for the PIM. It generates these reports using records from the transaction logs held by the TM. It uses predefined templates for generating audit and log reports. The application could bypass this mechanism and generate its own audit trails using the TM service. The audit trail generation is a local operation. There is no specific security requirements except for the service to function correctly and generate accurate audit trails. The logs are protected against tampering by intruders. All system operations log according to the audit level maintained in the GEPS environment.

All wallet operations are performed locally and do not need peer-to-peer communications. The security

requirements are integrity, confidentiality, and in cases where the machine is timeshared among multiple end-users, authentication. No non-repudiation service is necessary. The integrity and confidentiality of the wallet data can be ensured by signing and encrypting the data with the end-user's keys. The end-user must login in order to use the information. The login period can vary and depends on the application. Due to the security requirements and the locality of operations, the concepts of [IDUP-GSS-API] are valid and could be used. Namely, the wallet is considered an independent data unit and is protected using cryptographic operations.

#### 4.5.3 Value Transfer

The Value Transfer functionality refers to generic mechanisms to make a payment (positive transaction) from one account to another or to initiate a dispute (negative transaction). This is a peer-to-peer operation. Therefore, context maintenance is required.

The Value Transfer functionality is used during the Payment and Dispute phases, naturally. In addition, the Trace and Audit, Cancel, and Recovery phases may utilize some of its functionality.

The security requirements are integrity, authenticity, confidentiality and non-repudiation of message origin. The non-repudiation of message receipt is approximated through the use of signed receipts. The tokens generated and transmitted between parties are secured (signed and encrypted) meeting the above requirements.

#### 4.5.4 Error Handling

Error Handling is a critical functionality and is abstracted by the PIM for all of the other GEPS. This is used in all phases of operation. Therefore, applications have only one interface to deal with. As is, the PIM will provide a user interface to report errors and has code to act on resolving the errors. If a sophisticated application desires its own specialized error handling, it can bypass the PIM user interface while still using the underlying structures and classes (e.g. the ErrorTable).

All error handling functionality are performed locally and there is no need for peer-to-peer communications. The security requirements dictate that the integrity be preserved on the error reports and conditions. Since we assume that the operating system on which GEPS is run locally is trusted, there is no additional security mechanism used here.

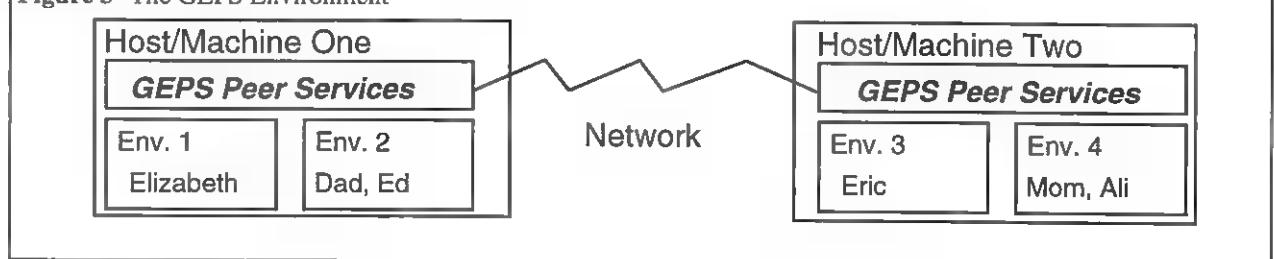
## 5 Implementation

We assume a single GEPS environment is set up by an administrator for every machine on the network. One or more end-users can share this environment. More than one application will use the same environment. If more than one application needs to change the environment and configuration at the same time, a locking mechanism must be used to ensure that data is not corrupted or lost. If more than one setup exists on a timesharing machine, a mechanism must be provided to allow end-users and applications to login and use their individual setup (e.g. mechanism which allows an end-user to login and have a home directory). The administrator performs some of the one-time operations involved in setting up the GEPS environment, including possibly installing payment systems and mechanisms to use. Figure 3 depicts the block diagram for the GEPS environment on a distributed network. The same GEPS on different environments (usually on different machines) are considered to be peers.

A Java implementation of GEPS is underway in our project—E-CO. The term E-CO stands for ECOlogy of E-COMmerce Services. In this project, we are focusing on public-key infrastructure and payment services. The main classes for each GEPS are provided in the Appendix. The detailed API specifications for our GEPS implementation will be provided in a separate document.

To better understand the use of GEPS by applications, we provide an example next. The applications have access to all GEPS functionality. However, the PIM simplifies this interface by providing a higher level of

**Figure 3- The GEPS Environment**



abstraction and a consistent user interface. Of course, application developers can choose to code with and without PIM. For example, specialized applications may desire to have their own wallet instead of that provided by PIM. Here, we use an example where the application is using PIM. This makes the illustration significantly simpler. This is only pseudo-code and is not depicting all of the programming details. Consider the case where an application is a web browser providing the end-user access to a storefront. The end-user is browsing the store and selecting items to purchase. The end-user then decides to pay. Once the payment is processed, the store provides the end-user with a receipt and processes the purchase and ships the goods to the end-user. Figure 4 illustrates the high-level code segment of the browser that would be necessary to implement the payment process.

To achieve transport independence, we have left the choice of communication mechanism to the GEPS implementations. For example, implementations may choose CORBA for this purpose (as is the case with Stanford's code [U-PAI]). Other communication requirements include the specification of the protocols between GEPS and infrastructure applications. While most existing protocols are specific to payment mechanisms, we anticipate that new communication protocols will be developed for

infrastructure applications such as those maintained by Banks or the Government.

## ■ Related Work

We have begun the implementation of the GEPS framework in the E-CO System project. This is done by first implementing the Payment Method Negotiation service. The result of our work is being published [PayNegot] as a companion paper. Additional information about our project can be found on our website at <http://eco.eit.com>.

Another effort, also dealing with payment negotiation, is the Joint Electronic Payment Initiative [JEPI] effort which we helped get started. The project, a joint effort between CommerceNet and W3C, will demonstrate a payment negotiation solution integrated with web-based purchasing. The difference between the JEPI and our MN implementation is that we have more closely integrated the MN with the other GEPS such as Preference Management.

In joint work with the Stanford Digital Library Initiative, we have further specified [U-PAI] parts of the GEPS framework by defining a mechanism-independent API to insulate the application programmer from the payment-specific details. The Stanford implementation uses CORBA to conduct

**Figure 4-** Example browser code fragment using GEPS

```
{
/* segment where payment related code begins */
InterfaceManager pim = new InterfaceManager()
pim.initialize()
PaymentContext context = pim.initiateContext()
complete = false;
while (not complete) {
    token=pim.Negotiate(context, total_price)
    send2peer(token)
    token = getResponse()
    if token.status = finished, then complete=true
}
complete = false;
pim.beginTransaction()
while (not complete) {
    token=pim.Payment(context, price of goods)
    send2peer(token)
    token = getResponse()
    if token.status = finished, then complete=true;
}
pim.commitTransaction(context)
pim.endTransaction(context)
} /* end payment related section */
/*
 * creates InterfaceManager */
/* PaymentWallet and PaymentConfiguration opened */
/* set up and maintain context for payment transaction */

/*
 * status lets application know when to stop */
/* application does not need to know content of token */
/* just send to peer (local function for message communication) */
/* just get response from peer (local function ) */
/* use price info during negotiation, asks end-user if necessary */

/*
 * begin atomic operation */
/* pay the amount using the negotiated capability */
/* just send to peer (local function for message communication) */
/* just get response from peer (local function ) */
/* use price in the payment from wallet */

/*
 * commit to the transaction to allow recovery */
/* complete transaction */
}
```

peer-to-peer communications. It is a partial implementation of the TM and CM and has some of the functionality of the PIM. While this is not a complete implementation of GEPS, many of the functionality have been incorporated. This work represents an approach to partially implement the GEPS taking advantage of its modularity.

JavaSoft's Java Electronic Commerce Framework [JECF] also represents a similar work to the GEPS framework, but with a Java-centric approach to building a software point-of-sale terminal accessible by Java-enabled browsers.

Another significant and related effort is the European Commission's activity called [SEMPER]. SEMPER is a European R&D project in the area of secure electronic commerce over open networks. While E-CO and SEMPER share many aspects of their vision, they differ in their implementation details. Similar to the GEPS model which we have presented here, SEMPER has a layered architecture. They identify a transfer layer in addition to a payment layer. The main component of the payment layer is called the Payment Manager. The Payment Manager includes some but not all of the functionality included in the GEPS. We have begun dialog with the members of the SEMPER project to find means for combining our respective work.

## 7 Conclusion

The excitement surrounding electronic commerce has created a lot of creative solutions for electronic payments. A solid framework with which to understand the various components and subtleties of this very complex problem helps us to not only understand the scope and magnitude of the problem, but also analyze the completeness of the proposed solutions. We believe that our layered approach and generic services provide such a framework. Our services support all phases of operation as specified in Section 3.4. The GEPS identified here represent a complete set with which all applications using electronic payments can be implemented. Only time will tell if the implementation of all the functionality is possible. However, based on our early findings and the result of our implementation so far, all indications are in favor of such achievements.

Our goal is to attempt to address the technical challenges introduced by the introduction of a variety of electronic payments solutions. There are, however, other non-technical hurdles one has to

overcome in order to successfully advocate the GEPS framework. These hurdles are related to the business acceptance of such a framework and the incorporation of such a framework into industry products. Currently, most businesses are engaged in competitive developments which prevents them from adopting open frameworks such as that advocated within this paper. Perhaps also, we are too early in the development stages of the electronic payments industry to know enough what the future will hold and what framework is the right encompassing open model. We are positioning the GEPS framework to address some of these issues, but only time will tell if we can achieve the industry acceptance required to make the framework ubiquitous.

## 8 Acknowledgments

The author would like to acknowledge the dedication and team work of the E-CO System project members without whom the implementations and many aspects of this work would not have been possible. They are Rajkumar Narayanaswamy, James Galvin, Andrea Stirrup, and Nick Zhang. Also discussions with the project members at Trusted Information Systems have been thought provoking and enlightening. They include Jeff Cook, Rich Feiertag, and Edward John Sebes.

The author has also worked closely for the past year with the Stanford Digital Library Initiative. Many thanks go to team members there for commenting on early versions of this paper. They are Hector Garcia-Molina, Scott Hassan, Steven Ketchpel, and Andreas Paepcke.

And in closing, many thanks belong to Donald Eastlake of CyberCash. The author worked closely with him while in the JEPI Design Team and has profited from many insights and fruitful discussions.

## References

[SET] See <http://www.mastercard.com/set/set.htm> or <http://www.visa.com/cgi-bin/vee/sf/set/intro.html>.

[WebSite] See E-CO System Project website at <http://eco.eit.com>.

[PayNegot] Alireza Bahreman and Rajkumar Narayanaswamy, "Payment Method Negotiation Service". *Proceedings of the Second USENIX Electronic Commerce Workshop*, November 1996.

[U-PAI] Steven Ketchpel, Hector Garcia-Molina, Andreas Paepcke, Scott Hassan, and Steve Cousins, "U-PAI: A Universal Payment Application Interface".

*Proceedings of the Second USENIX Electronic Commerce Workshop*, November 1996. See also:  
<http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1996-0039>.

[NetBill] Benjamin Cox, J. D. Tygar, and Marvin Sirbu, "NetBill Security and Transaction Protocol" in *Proceedings of the First USENIX Workshop on Electronic Commerce*, July 11-12, 1995.

[NetCheque] See <http://gost.isi.edu/info/netcash/> and <http://gost.isi.edu/info/netcheque/>.

[CyberCash] D. Eastlake 3rd, B. Boesch, S. Crocker and M. Yesil, "CyberCash Credit Card Protocol Version 0.8". Internet Request for Comment, RFC 1898, February 1996. See also:  
<http://www.cybercash.com>.

[GlobeOnline] Paul Andre Pays and Fabrice de Comarmond, "An Intermediation and payment system technology". Computer Networks and ISDN Systems November 28 (1996) 1197-1206. Available online at [http://www5conf.inria.fr/fich\\_html/papers/P27/Overview.html](http://www5conf.inria.fr/fich_html/papers/P27/Overview.html). See also <http://www.gctech.com>.

[PKI-API] Jeff Cook, and John Sebes, "A Generic API for PKI Services". Available from the E-CO website.

[GSS-API] J. Linn, "The Generic Security Service Application Programming Interface (GSS-API)". Internet Request for Comment, RFC 1805, June 1995.

[IDUP-GSS-API] C. Adams, "Independent Data Unit Protection Generic Security Service Application Programming Interface". Internet Draft <draft-ietf-cat-idup-gss-05.txt>, June 11, 1996.

[JEPI] See <http://www.w3.org/pub/WWW/Payments> or <http://www.commerce.net/pr/041796.jepi.html>.

[JECF] See <http://java.sun.com/products/commerce>.

[SEMPER] See <http://www.semper.org>.

## Contact Information

You can reach the author at:

Alireza Bahreman  
bahreman@eit.com  
EIT/VeriFone, 530 Lytton Avenue  
Palo Alto, CA 94301-1539, USA  
TEL: +1 415-463-1232  
FAX: +1 415-617-9746

## Appendix - The GEPS API Classes, Draft Version 0.1

Please note that this is a work in progress and therefore, the specifications might change. The classes in **Bold** typeface are primary objects of individuals generic electronic payments services.

**Table A** - The High-level GEPS API Classes and their Primary Functions—Payment Interface Management

<i>CLASSES</i>	<i>Primary Functions</i>
<b>InterfaceManger</b>	Provides an abstraction to programmers and end-users for using the Generic Electronic Payment Services.
PaymentConfiguration	Maintains configuration settings and provides information on the environment and infrastructure applications for all services. Contains several sections, at least one for each GEPS (MN, PM, CM, and TM).
PaymentWallet	Maintains bindings between payment capabilities and individual account holder information.
PaymentInstruction	Provides a generic template for messages that convey an instruction to transfer value from one account to another.
PaymentDispute	Provides a generic template for messages that convey a request for return or credit against an existing payment transaction.
PaymentContext	Maintains the payment transaction context and is used to relate and synchronize multiple messages for the same transaction.
PaymentToken	A generic message to pass between PIM peers which conveys the functions to be performed in a payment transaction. The application must transport them between peers.
ErrorTable	Catches all error signals or exceptions from GEPS and provides a unifying interface to the application.

**Table B** - The High-level GEPS API Classes and their Primary Functions—Payment Method Negotiation

<i>CLASSES</i>	<i>Primary Functions</i>
<b>PaymentNegotiator</b>	Handles the payment negotiation process.
IOProcessor	Handles the wrapping and unwrapping of tokens for transmission between peers. Provides transport independence by allowing subclassing for different transport mechanisms.
UserInterface	Interface specifying the main elements any user interface must have to perform the selection process during negotiation with the end-user. Can be subclassed for specialization.
NegotiationContext	Maintains the negotiation context on a per transaction basis for the negotiating peers.
NegotiationToken	A generic message to pass between MN peers which conveys the functions to be performed for the negotiation. The application must transport them between peers.

**Table C** - The High-level GEPS API Classes and their Primary Functions—Preference Management

<i>CLASSES</i>	<i>Primary Functions</i>
<b>PreferenceManager</b>	Provides management and maintenance for the end-user's references with respect to payment mechanism usage and spending limits.
PaymentPreference	A record of the end-user preference.
PreferenceCriteria	A record of the criterion under which the preference is being made. Example criteria include cost of transaction, spending limits, transaction value, and brand identity.

**Table D - The High-level GEPS API Classes and their Primary Functions—Capability Management**

<i>CLASSES</i>	<i>Primary Functions</i>
<b>CapabilityManager</b>	Provides the necessary glue to “plug-in” payment systems to the GEPS. Also ensures a smooth transition to and from payments.
PaymentCapability	An object representation of a payment mechanism or system.
Account	A representation of a “real” account with value.
AccountProxy	An online representation of a physical account. The AccountProxy does not hold real value.
ValueToken	A representation of real value. Examples are electronic cash. If lost, the value represented by them is also lost.
CapabilityContext	Used to maintain context during Inquiry or Payment/Dispute operations on accounts.
CapabilityToken	A generic message to pass to the payment system peer.

**Table E - The High-level GEPS API Classes and their Primary Functions—Transaction Management**

<i>CLASSES</i>	<i>Primary Functions</i>
<b>TransactionManager</b>	Provides management and maintenance of the transaction logs.
<b>TransactionRecovery</b>	Provides transaction recovery capabilities.
<b>TransactionAuditor</b>	Enables audit and trace on the contents of the log using flexible reporting mechanism that can be adapted to application specific needs.
PaymentTransaction	Transaction record for positive usage or payment.
ReceiptTransaction	Transaction record for an acknowledgment or receipt on payment.
DisputeTransaction	Transaction record for negative usage or dispute.
NegotiationTransaction	Transaction record for negotiation process.
PreferenceTransaction	Transaction record for preference process.
CapabilityTransaction	Transaction record for capability process.



# U-PAI: A Universal Payment Application Interface \*

Steven P. Ketchpel, Hector Garcia-Molina, Andreas Paepcke, Scott Hassan, Steve Cousins  
Stanford University  
Computer Science Department  
Stanford, CA 94305  
{ketchpel, hector, paepcke, hassan, cousins}@cs.stanford.edu

## Abstract

The progress of electronic commerce has been stymied by the lack of widely accepted network payment mechanisms. A number of proposals have been put forward, and each one offers a slightly different protocol and set of features. Yet none has achieved the critical mass to become an accepted standard. We believe that there will continue to be a variety of payment mechanisms, so in this paper we propose U-PAI, a universal payment application interface that will enable a programmer to write for one interface, and then interact with any payment mechanism. Each payment mechanism can support the universal API directly, or a *proxy* or wrapper can be built to translate U-PAI calls to the appropriate native calls supported by the payment mechanisms. In this paper we illustrate how two such proxies could be built. We also provide, in the appendix, a full CORBA specification of U-PAI.

## 1 Introduction

A payment mechanism is a means by which economic value is transferred between two parties, possibly using some intermediaries. It should be secure, easy to use, and have low transaction costs. Even though all electronic payment mechanisms have these same goals, there are many variations between mechanisms, (see for example, [9, 4, 7, 6]). Some of the variations can be minor, e.g., the order or nature of parameters in a function call. Other differences are more substantial, such as using different transport mechanisms and protocols like HTTP, telnet or e-

mail. The most significant difference, however, is the order of steps required to execute a payment. One payment mechanism, Millicent[7], requires the payer to acquire "scrip" from a broker before an interaction, while a second, the anonymous credit card[6], channels all communications through a re-mailer to keep identities hidden. If a merchant wants to support several payment mechanisms, not only must the merchant have accounts with each, but he or she must also tailor the application software to determine which mechanism is in use by the customer and generate the proper payment protocol steps to the customer and intermediaries.

The diversity of payment mechanisms may be beneficial in the long run because it encourages competition and enables an exploration of a broader space of solutions. However, this diversity is also a significant barrier to commerce: customers must maintain accounts with several different payment mechanisms. Furthermore, merchants and customers both find that there is no standard way for payment mechanisms to interact with application software such as a browser or electronic storefront.

Our goal in this paper is not to add to the diversity by introducing another payment mechanism, but rather to define a common set of functions that act as a layer of abstraction between application software and payment mechanisms. This Universal Payment Application Interface (U-PAI) will ease the burden on software developers at both the consumer and merchant level. Merchants and customers do not need to customize their applications to support each individual payment mechanism, since an application supporting this one universal API will interact with a broad range of payment mechanisms.

We hope that the benefits of standardization will encourage payment mechanism providers to support U-PAI (perhaps in addition to their own API which provides additional or different functionality). However, we recognize that payment systems providers see their proprietary protocols as a differentiating fac-

\*This material is based upon work supported by the National Science Foundation under Cooperative Agreement IRI-9411306. Funding for this cooperative agreement is also provided by DARPA, NASA, and the industrial partners of the Stanford Digital Libraries Project. The first author was partially supported by a National Defense Science and Engineering Graduate Fellowship.

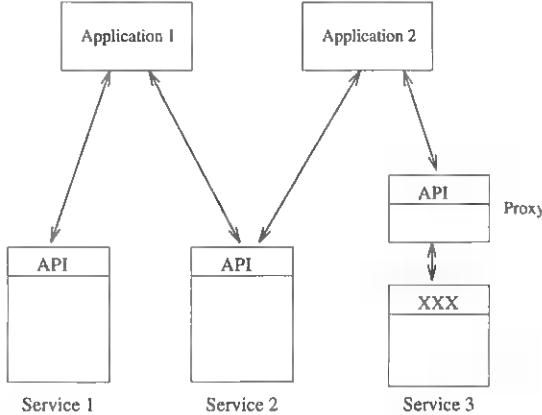


Figure 1: Universal Payment Application Interface abstracts payment mechanism internals

tor and way to retain market share. One approach to achieve widespread use of the U-PAI protocol would be to propose it to the relevant standards bodies and proceed through the ratification process.

An alternative approach is to build *proxies* or wrappers or gateways to popular payment mechanisms, as illustrated in Figure 1. Each proxy translates U-PAI calls into native calls to the underlying payment mechanism. This notion of proxy is widely used when accessing heterogeneous resources, be they databases, search engines, or other services [8]. By developing proxies and distributing them freely for the most popular payment mechanisms, we can encourage application developers to experience the benefits of using a single protocol, which may result in their reluctance to devote implementation effort to systems which do not support the protocol. This type of pressure may effectively encourage other payment systems providers to support the interface.

Of course, since each payment mechanism offers different features, it is impossible for a single API to capture all of the functionality of all of the mechanisms. Thus, the challenge we face in designing the common API is to identify the essential features that are used in the vast majority of interactions. A second challenge is to design, for these common features, an elegant interface that simplifies the programming task. One significant aspect of this challenge is that important steps need to be asynchronous, non-blocking calls. Asynchrony permits multiple payments to be in process at the same time, or may allow a payment to be aborted after it has been authorized, but before it has been completed. We believe that U-PAI meets those goals.

Having defined U-PAI, the next challenge is to show that one can build proxies to existing payment

mechanisms, and that these proxies can support the necessary common functionality even if the underlying mechanism uses a different payment model or ordering of steps. We have studied a number of existing mechanisms and shown how U-PAI can support the basic functionality of all these schemes. We will illustrate two such proxies, one supporting First Virtual, the second, DigiCash's ecash product.

In the following section we describe some related work. Section 3 defines all of the functions which are part of the interface. Section 4 shows a sample transaction, giving each step from start to finish. Section 5 shows how a proxy might be constructed for the First Virtual payment mechanism. An equivalent ecash proxy appears in Section 6. In Section 7, the case of failed transactions is considered in greater detail, along with security concerns. Finally, Section 8 offers a summary. The full CORBA specification in ISL, the interface specification language of Xerox PARC's ILU, appears in the Appendix.

## 2 Related Work

It is important to note that payment mechanisms and U-PAI are only one part of a larger electronic commerce environment. U-PAI only covers the basic functionality of accounts and payments, e.g., checking the balance of an account or transferring funds from one account to another. It does not cover price negotiation, return of defective goods, bidding, and other commerce issues. These require other API's that would work in conjunction with U-PAI. A broader view of the issues related to electronic payments may be found in [1], which presents the Generic Electronic Payment Services framework. Of the five modules discussed there, U-PAI performs "Capability Management" tasks, some of which may appear in the higher level "Payment Interface Manager".

Another attempt to address the diversity of payment mechanisms is the Joint Electronic Payment Initiative (JEPI) project, co-sponsored by CommerceNet and W3C. The focus of JEPI is reaching agreement between the customer and merchant on a payment mechanism[2]. JEPI is built on top of East-lake's Universal Payment Preamble (UPP)[5]. Neither of these systems achieves the level of integration that is proposed in U-PAI. Application developers must still implement a different protocol for all of the payment mechanisms they choose to support; JEPI and UPP merely allow the customer and merchant to select the protocol that a particular transaction will use. Therefore, it would be possible to employ JEPI to select a payment mechanism and then use U-PAI

to control the processing within that mechanism.

Finally, we note that our work is being done in the context of the Stanford Digital Libraries Project, where we are studying how to provide access to the resources and services being developed under the NSF/DARPA/NASA Digital Libraries Initiative (see <http://www.dlib.org/projects.html>). Clearly, payment is one of the central issues in such an environment. This work was performed in collaboration with EIT/VeriFone, under the auspices of CommerceNet (see <http://www.commerce.net/>), where again, facilitating interactions among customers and merchants with different payment mechanisms is crucial.

### 3 API Definition

U-PAI was designed from an object-oriented point of view. The interface offers a set of active objects, with their associated methods. Making a call to U-PAI involves calling a method on one of these objects. Similarly, the interface specifies certain objects that the application is expected to have that can be called by U-PAI, for example, to notify the application when a payment transaction terminates.

The equivalent functionality of the interface can be captured through non-object-oriented means as well. Entities which are objects in the API would be construed as records, with the object ID representing an identifying index for the record. Method invocations are replaced with a remote procedure call that passes the record which is to be acted upon as an explicit parameter of the call. In the interest of clear presentation, the object-oriented method will be used throughout the rest of the exposition.

In this section we describe the main object types in the API and their methods. Some of the methods are used to access what conceptually are “internal fields” of the object. For example, as we will see later, a payment control record (**PCR**) named **P** has an **Amount** field that gives the amount of money being paid. This value can be read by invoking **P.GetAmount()** and can be set by **P.SetAmount()**. In reality, **P** may not have a field with this value (in which case we say it is not *materialized*), but **P.GetAmount()** may invoke a function to compute the amount based on other internal or external information. However, for understanding the interface, it is useful to think of **Amount** as a field in **P**. Also, note that often the **Set** method will be disabled for some fields, e.g., the application may not set the balance of an account. Thus, to describe each of our objects, we first define their “fields” and then other methods they may have. (Full formal

definitions may be found in the Appendix.)

#### 3.1 Account Handles

An **AccountHandle** instance is a representation of a real-world account. For example, a user may have several VISA **AccountHandles**, corresponding to the cards issued by different merchant banks belonging to the VISA network. The user creates an **AccountHandle** when he wishes to start making electronic payments with the account. He may query balance and credit limits on the account by making appropriate calls on the **AccountHandle** object.

A helpful analogy to clarify the notion of accounts and **AccountHandles** is that of UNIX files (see Figure 2). A file can be created and deleted, which corresponds to the creation and closing of a real world account. When the file exists, it is possible for a program to reference it by opening the file, making read and write accesses to it, and closing the file. In the payments world, this corresponds to generating an **AccountHandle**, making transfers, and erasing the **AccountHandle**. The real world account continues to exist even after the electronic **AccountHandle** representation has been deleted, just as a UNIX file exists after a program referencing it closes the file and deletes the file handle.

Conceptually, an **AccountHandle**, **ah**, has the following internal fields, although as noted below, some of them may not be actually materialized.

- **Balance**: This is the amount of available money available at **ah**'s account for payments. A positive amount indicates that the account holder has a positive stored balance. For example, Digi-Cash's ecash would be represented as a positive balance, since the user has already “purchased” the ecash. In contrast, a negative amount indicates the account owner owes money. Charges against a credit card would result in a negative balance that would be brought (closer) to zero when a payment was made to the card issuer. A query to this field, via **ah.GetBalance()**, will often require a real-time query to the account issuer, such as the bank that issued a MasterCard, in order to determine if non-electronic payments have been made. (In this case, we say that **Balance** is not materialized at **ah**.)
- **CreditLimit**: This is the amount of credit that may be charged on a credit-based payment mechanism. It is a negative value, and the balance on an account should never go below it. For non-credit instruments, its value is zero.

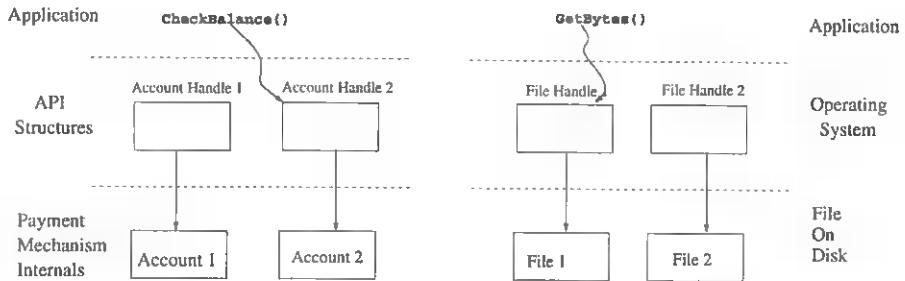


Figure 2: The similarities between an account handle and a file handle

- **AccountType:** This is an identifier (ID) of the type of account, e.g., First Virtual, VISA/SET, DigiCash, and so on. The value is of type `AccountTypeID`.
- **TransferAccountTypesFrom:** This is a list of `AccountTypeID`'s that this account can receive transfers from. So, for instance, a Mark Twain Bank ecash account can receive transfers either from another ecash account, or from the account holder's checking account.
- **TransferAccountTypesTo:** This is a list of `AccountTypeID`'s to which this account can make transfers.
- **MechanismProperties:** This is a *property set* that includes descriptive traits of the payment mechanism used by this account. Each entry in the property set is a property name and value. These properties assist the user in choosing which payment mechanism to use for a particular transaction. The name of the payment mechanism is stored in the string property `name`. The `fixed-cost` property is an amount which describes the fixed portion of the overhead cost for using this payment mechanism. The `percentage-fee` property records the variable cost. The expected time for one payment may be found in the `time` property. The boolean property `anonymous` records whether payments made using this mechanism may be linked to the user. Any other property may be added at the discretion of the payment system provider.

`AccountHandles` are typically subclassed with the specific type of the payment mechanism. The interface is inherited from the base class `AccountHandle`, but the methods are overridden with the specific details appropriate for that payment mechanism. For example, if a user wanted to create an `AccountHandle` for his First Virtual (FV) account, he would create an instance of a `FVAccountHandle`,

which is in turn a subclass of `AccountHandle`. `FVAccountHandle` must have all the methods of an `AccountHandle` (though they will be implemented in a way idiosyncratic to First Virtual).

The following methods can also be invoked on an `AccountHandle`, `ah`:

- **OpenAccount(PropertySet acctinfo): Any**  
Typically, an application first creates a new `AccountHandle` object `ah` and then invokes `OpenAccount` on it to "initialize" `ah` to identify the appropriate real world account. The `acctinfo` parameter contains the necessary information to identify the real world account. The parameter is a property set that associates arbitrary field names with different types of parameter objects. For example, if we are opening a VISA account, `accountinfo` may contain the associations "type: VISA", "account: 123-456-789" and "expiration: 03/99." If we are opening a First Virtual account, we may need "type: FV", "name: John Doe" and "email: doe@whitehouse.gov." This method will then set up the `AccountHandle` as indicated, for instance, it will initialize field `TransferAccountTypesFrom` to indicate what type of account this FV account can receive funds from. We stress that this process does not establish a new FV account, it merely creates a representation of an existing FV account so that it may be used for payments through U-PAI calls. The return value of opening an account may be used as a security/authorization token to allow the object creator to identify itself to the account in the future.

- **CreateAccount(PropertySet acctinfo): Any**  
This method creates a new real world account using `acctinfo` and updates the internal fields of `ah` to refer to it. Not all payment mechanisms will offer the option of creating a real world account through purely electronic means invoked

remotely by a user. The return value may be used for authorization.

- **CloseAccount()**

This method deletes the handle `ah`. The underlying real world account is unaffected. Future references to `ah` will result in an error.

- **DeleteAccount()**

This method deletes both the `AccountHandle` and the real world account which it represents. Again, not all payment mechanisms will support this method.

- **GetStatus(RefIDType Ref):PaymentStatus**

This method provides direct access to the payment mechanism's records concerning a particular transaction. In the event that the `PCR` (described next) is unavailable, an alternative (though probably more expensive) entry point exists. Not all payment mechanisms will support this method.

## 3.2 Payment Control Records

A *Payment Control Record* (`PCR`) instance is a representation of a single payment transaction. An application creates a new `PCR` for each individual transfer between two accounts. The `PCR` is then the locus of control for all activities regarding that payment.

Conceptually, a `PCR`, `p`, has the following fields:

- **RefID:** Provides a unique identifier for this payment. The value is of type `RefIDType`.
- **ContextID:** Identifies the context for this payment. The value, of type `RefIDType`, contains application specific information such as the invoice for which this payment is being made.
- **Amount:** the amount of money that is being paid by `p`.
- **DestAccountHandle:** Identifies the account receiving the funds.
- **DestAccountAuthorization:** Conveys the authority to deposit money in the destination account.
- **SourceAccountHandle:** Identifies the account supplying the funds.
- **SourceAccountAuthorization:** Conveys the authority to withdraw money from the source account.

- **Receipts:** Information, such as a receipt or decrypting key, that is given to `p` at the start of the transfer, and should be revealed to all participants upon successful completion. The payment mechanism may add additional receipts to this field.

- **Status:** The status of `p` is a list of entries representing the history of this payment. Each entry is made up of two components, a `MajorStatus`, which takes one of three values (`PaymentComplete`, `InProgress`, or `Failed`), and a `MinorStatus`, which provides greater detail about the current status. The entries are ordered with the most recent appearing at the front of the list.

Applications may make use of these values, though in many cases, the values will be payment-mechanism specific. Some sample entry values are shown in Table 1.

- **MonitorList:** `Monitor` objects (described in the next section) provide a way for applications to request notifications of status changes, rather than directly poll the status of `p` through the method `p.GetStatus()`. The `MonitorList` field of `p` is a list of `Monitor` objects that must be notified when the status of `p` changes.

To perform a payment, an application creates a `PCR` object, call it `p`, with the information that describes the desired operation. Then it invokes methods on the object to start or abort the payment.

- **StartTransfer()**

This method initiates the transfer of funds in order to effect the transfer described in `PCR p`. This call is non-blocking, so that customer processing may continue even before the payment has completed. A transfer requires authorization from both account holders to withdraw funds from one account and deposit them in another. This method should be invoked only once per `PCR`.

- **TryToAbortTransfer()**

This function attempts to abort a transaction which was previously initiated by a `StartTransfer`. The payment may have been already completed, or reached some commit point so that it is too late to abort. Feedback is given to the calling application only via the status of the `PCR` and the `Monitor` objects.

- **UpdateStatus(StatusEntry stat)**

This last method is invoked by whatever entity

Table 1: Payment Transaction Status Values

MajorStatus	MinorStatus	Description
PaymentComplete		Money transferred from payer to payee
InProgress		Transfer started, not completed
Failed	Aborted	The payment was aborted
Failed	NotSufficientFunds	Not Sufficient Funds for payer to make payment
Failed	NoSourceAccountSelected	The AccountHandle has not yet been associated with an account by <code>create()</code> or <code>open()</code> .
Failed	UnauthorizedSourceAccount	Payer not authorized to make payments from this account
Failed	UnauthorizedDestAccount	Payer not authorized to make deposits to this account
Failed	NonExistentDestinationAccount	Payee account not recognized
Failed	UnabletoTransfertoAccountType	Payee account wrong type

is actually performing the payment transaction to report a change in status. Parameter `stat` is a **MajorStatus**, **MinorStatus** pair which is appended to `p`'s `status` field.

Incidentally, in some cases an application may wish to make more than one payment to cover a single invoice. For example, having received a bill for some delivered good, the application may wish to pay half of the amount due with a credit card and the other half with a check. In this case, the application creates two separate PCR's, each with the appropriate amount to pay. In this case, each record could have the same `ContextID` field since the same invoice is involved.

### 3.3 Monitors

A **Monitor** instance is an object used to supplement the status tracking feature of a **PCR**. Rather than requiring the application to routinely poll the status of the **PCR**, the application programmer may choose to implement a **Monitor** object which receives notifications whenever the payment mechanism updates the status of the **PCR**. Several such **Monitor** instances may be active at any time. For example, monitors acting on behalf of the payer and payee (and any other parties to the transactions, such as a state tax board) act as the recipients of messages which the **PCR** re-broadcasts as the transfer proceeds. For instance, if a bank refuses a check due to insufficient funds, the **PCR** reflects a failed status, and passes that information to each active **Monitor**. **Monitor** objects are written by the application programmer, providing the linkage between the result of the payment mechanism and the desired behavior of the application.

A **Monitor** object, `m`, conceptually has a `status` field just like a **PCR**. The following method can be invoked on `m` to update the field:

- **Notify(PCR p, StatusEntry s)**

This function updates the record of the transaction's status as recorded at `m`. In practice, this method also performs application specific tasks, depending on the nature of the notification.

In many cases, this basic **Monitor** class will be subclassed by the application programmer to provide additional functionality. For instance, one common usage will be to provide monitors with timeout capabilities. In this case the subclass may add methods such as `register` to define a timeout and `unregister` to cancel it. If a timeout occurs without the payment completing successfully, then the monitor can automatically attempt to abort the payment.

### 3.4 Additional Payment Functions

From the point of view of the application, a payment is initiated by calling on method `StartTransfer` of the appropriate **PCR**, say for instance, `p`. This is natural since the **PCR** is the locus of control for the payment. However, it is difficult for a method in a generic payment record to actually execute the transaction, since it depends on the specific account types involved. We solve this problem by having `p.StartTransfer()` call a method `ah.StartTransfer(p)`, where `ah` is the source `AccountHandle`, i.e., where `ah = p.GetSourceAccountHandle()`. This latter method then actually makes the necessary calls to the underlying payment mechanism(s).

In summary, the following two methods of an **AccountHandle** can be invoked by the system, but should not be by the application programmer:

- **StartTransfer(PCR p)**

This method is invoked by the PCR. All transfers should be initiated through the **StartTransfer** method of the PCR.

- **TryToAbortTransfer(PCR p)**

This method is invoked by the PCR. All transfers should be aborted through the **TryToAbortTransfer** method of the PCR.

## 4 Sample Transaction

In this section, we will walk through the steps required for a typical transaction. These involve:

1. Creating an **AccountHandle** (done once)
2. Creating a **Monitor** object (done once or once per transaction)
3. Creating the PCR (Payment Control Record)
4. Initiating the transfer at the PCR
5. Initiating the transfer at the **AccountHandle**
6. Updating the status at the PCR
7. Calling back to the **Monitor** object

This transaction will be a typical “mail-order” one, with the merchant dictating the terms of the purchase, the customer placing an order and sending payment, and finally, the merchant sending the goods.

### 4.1 Creating an AccountHandle

In this example, the customer wishes to enable his First Virtual account to make payments within this system. His FV account has the account identifier “jsmith”. There is a subclass of **AccountHandle**, called **FVAccountHandle**, for creating First Virtual accounts (developed by the people at First Virtual or a third party proxy-generator).

```

■ FVAccountHandle jmsFVacchandle;
(* This creates a new object *)

■ FVAuth =
  jmsFVacchandle.OpenAccount({"type:
  FV", "user-id: jsmith", "e-mail:
  jsmith@nowhere.net"})

```

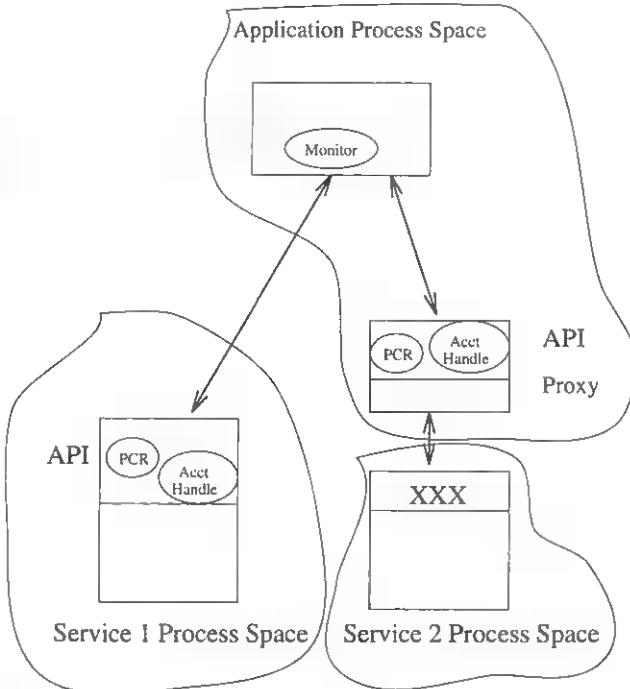


Figure 3: Location of system components in the system.

This **jmsFVacchandle** object is a representation of the First Virtual account in the payment system. Its type is **FVAccountHandle**. The method implementations were written by the First Virtual development staff or proxy writers, but this object is now customized with J. Smith’s account information. Further messages to it will result in communication with the First Virtual system to perform the desired operation and may rely on the return value of the **OpenAccount** method to provide authentication. The **AccountHandle** is located on the payment service side, or at the proxy, which may be running locally on the customer’s machine. Figure 3 shows the location of key components in the distributed system.

### 4.2 Create a Monitor object

The buyer needs to have some method of keeping track of the status of various transactions. The **Monitor** object performs this role, located on the customer’s machine, receiving updates as to the transaction status, and triggering application actions accordingly. For instance, if the payment is complete, the **Monitor** object should set up a process to receive the goods or complain if they are not received in a timely fashion. If payment stalls due to a problem such as insufficient funds, the **Monitor** object should

choose an alternative payment mechanism if it is so authorized, or alert the application program to the problem.

In this example, we assume that the buyer will create a **Monitor** object exclusively for this transaction. The application programmer has developed a **CustomerMonitor** class which inherits from the **Monitor** object of U-PAI. The **CustomerMonitor** must support the one method of a **Monitor** object, **Notify**. The implementation details are application specific. A small piece of a typical monitor's **Notify** method is given here:

```
Notify(PCR whom, StatusEntry s):
```

```
if s.MajorStatus == PaymentComplete:  
    DeliveryMon.expect(whom.GetInvoice())  
elseif (s.MajorStatus == Failed) &&  
    s.MinorStatus == "NotSufficientFunds":  
    self.SelectNewPaymentMech(whom)
```

The **StatusEntry** record has a **MajorStatus** field of enumerated type (**PaymentComplete**, **InProgress**, or **Failed**) and a **MinorStatus** which provides additional detail about the update. The **Monitor** object is responsible for determining what to do based on this new status. In the fragment above, it calls the application specific **DeliveryMon** if payment is complete, or tries to select a new payment mechanism if this one failed due to insufficient funds. These routines are both outside the scope of U-PAI.

The new **CustomerMonitor** object (fulfilling the role of the **Monitor** object) is created in the declarations before the transfer is started.

```
■ CustomerMonitor CM;
```

The **Monitor** object should, at a minimum, support actions for each of the three **MajorStatus** values. If the application programmer knows in advance about specific payment mechanisms that will be used and the status values that they report (through the **MinorStatus** descriptions), the application can use this information in determining what step to take next.

### 4.3 Creating the PCR (Payment Control Record)

By creating a **PCR**, the application tells the payment component how much money should be sent to whom, from which account, and how to inform the application and other interested parties of updates. In this example, the customer has obtained the merchant's **FVAccountHandle**, authorization to deposit

into that account, and a **Monitor** object (probably from an invoice or advertisement provided by the merchant) and has stored them in application variables **MerchantAcctHandle**, **MerchantAuth** and **MerchantMonitors** respectively. The amount that the customer intends to pay is \$4.00. The authorization code to use this source **FVAccountHandle** was generated by the **OpenAccount** method from Section 4.1 and was stored in **FVAuth**. Neither customer nor merchant needs to create a receipt for this transaction. The reference number is XE-2909, and is a payment for invoice number AXP-309. Due to the close relation between the **PCR** and the payment, the **PCR** is also typically located at the server or proxy.

- **PCR** pay;  
 (\*Creates the pay object of PCR type\*)
- **pay**.SetDestAccountHandle  
 (**MerchantAcctHandle**)
- **pay**.SetDestAuthorization(**MerchantAuth**)
- **pay**.SetSourceAccountHandle  
 (**jmsFVacchandle**)
- **pay**.SetSourceAuthorization(**FVAuth**)
- **pay**.SetMonitorList(**MerchantMonitors** ∪  
 {CM})
- **pay**.SetReceipts([])
- **pay**.SetRefID("XE-2909")
- **pay**.SetContextID("AXP-309")
- **pay**.SetAmount(4.00, "USD")

### 4.4 Initiating the transfer at the PCR

Once the buyer has completed the **PCR** object, he is ready to make a payment, and only one command is necessary.

```
■ pay.StartTransfer()
```

If the buyer has filled in the **SourceAuthorization** field, anyone, including the merchant, can invoke the **StartTransfer** method. Once the **PCR** receives a request to initiate the payment, it passes it through to the **AccountHandle**, which has the appropriate payment mechanism-specific code to continue the operation.

#### 4.5 Initiating the transfer at the AccountHandle

The PCR notifies the Monitor objects that the payment has been initiated, and they should expect additional updates on it. Then it interacts with the payment mechanism to accomplish the funds transfer. In this case, the **AccountHandle**, acting as a proxy, mimics the First Virtual protocol, generating, sending, receiving and processing e-mail messages (details in Section 5). Status updates will be sent to each Monitor object mentioned in the PCR's **MonitorList**, namely the **CustomerMonitor** instance CM and each Monitor object that the merchant supplied.

```
■ for m in self.MonitorList:  
    m.Notify(self, [InProgress, "Payment  
Initiated"])  
  
■ (self.getSourceAccountHandle()).  
StartTransfer(self);
```

#### 4.6 Updating the status of the PCR

As the payment mechanism progresses through the steps of its internal process, it may periodically issue status updates to the PCR. It does this by means of the **UpdateStatus** method, invoked on the PCR. The PCR passes its own identity to the Monitor objects so that they can distinguish among the multiple transactions they may be monitoring. The payment mechanism interacts with the **AccountHandle** to trigger the status updates in the rest of the payment system.

```
■ thisPCR.UpdateStatus([PaymentComplete,  
"termination normal"])
```

The identifier **thisPCR** is set to the PCR that was passed to the **AccountHandle** in the **StartTransfer** call. A payment mechanism may issue as many **InProgress** updates as it wishes, each with a different **MinorStatus** value. The payment mechanism or its proxy must make the **UpdateStatus** call when the payment terminates, either successfully or unsuccessfully.

#### 4.7 Calling Back to the Monitor Objects

When the PCR receives a status update, it is responsible for echoing that update to each Monitor in its **MonitorList** field. When the transaction is successfully completed, the PCR's **Receipts** field is broadcasted to the monitors.

```
■ for m in self.MonitorList:  
    m.Notify(self, [PaymentComplete,  
    self.Receipts])
```

The application may need to map back to the payment details of this transfer by getting its associated context object (such as the invoice), by invoking the **GetContextID** method on the PCR. When the Monitor object learns that the payment has completed, it takes the application specific behavior dictated in the **Notify** method. In our example, that involves calling the **DeliveryMonitor** to await the arrival of the ordered goods. The merchant's Monitor object would initiate the delivery of the order.

The transaction continues with the payment mechanism possibly making several **UpdateStatus** calls which are re-broadcast as **Notify** to the monitor list, until eventually the transaction completes with either a **Failed** or **PaymentComplete** status. At that time, the PCR is still accessible to the application, if it wishes to **GetStatus**, or the application may deallocate the space (garbage collect) the PCR.

### 5 Sample First Virtual Proxy

In this section, we show how one real world payment system can support this API without changing its current operation. The First Virtual (FV) payment mechanism (see <http://www.fv.com/>) was the first service which allowed consumers to transfer real money across the network, requiring both payer and payee to hold FV accounts. It works by assigning each user a new account name, and obtaining the user's credit card information in a secure, out-of-band channel. Designed primarily for information goods that merchants can produce and distribute for effectively zero marginal cost, the FV management encourages its merchants to give consumers a chance to "try before you buy", with the opportunity to refuse payment for the goods.

The full structure of a FV transaction (see Figure 4) is:

1. The customer sends his FV account information to the seller via e-mail.
2. The seller can optionally verify the existence of the account with FV, again by e-mail (optional, not shown in figure).
3. The seller delivers the goods to the buyer's e-mail address (which should match that of the FV account). This step is outside the scope of the payment process, and not shown in the figure.

4. The seller sends a charge request (via e-mail or telnet) to FV asking FV to bill the buyer.
5. FV sends an invoice to the holder of the FV account via e-mail.
6. The buyer responds by e-mail either indicating that he accepts the charge, acknowledges requesting the merchandise but does not want to pay for it, or does not recognize the charge and suspects fraud.
7. FV updates account balances if payment was approved, and informs the merchant of the resolution, using e-mail.

Some time later, FV aggregates the charges made by the user into a single charge to be levied on the associated credit card and paid to FV. Some time much later (90 days), the money is deposited in the appropriate merchant's checking account.

Figure 5 shows how the FV payment mechanism could interact with U-PAI. Steps labeled "A", "B", "C", and "D" correspond to the steps described in Sections 4.4, 4.5, 4.6, and 4.7 respectively. We assume for the sake of this example that the creation of the **AccountHandle** and the **Monitor** have been completed already. The application begins by creating a **PCR** (producing the object labeled as such in the figure) and then initiating a fund transfer (Step A), on the new **PCR**. In Step B, the **PCR** re-directs the call to the **AccountHandle** which acts as a proxy for First Virtual, receiving U-PAI messages, then translating them into the e-mail forms which are required by the First Virtual process. Forming another part of the FV proxy, the merchant's **AccountHandle** intercepts this mail message (Fig. 5, Step 1) and forms a FV invoice, which is sent to the FV commerce server (Step 4), possibly issuing a status update to the **PCR** as well. The FV service, ignorant that the e-mail invoice was automatically generated by the proxy, proceeds as it would if the invoice had come from a human, sending a copy of the invoice on to the specified customer, asking for approval (Step 5). Here part of the FV proxy working on the payer's machine intercepts the mail message and (assuming no **TryToAbortTransfer** invocation has been made) sends its approval to the FV server (Step 6), again with a possible update to the **PCR**. The FV server once again completes the processing, actually transfers the money, and sends the merchant e-mail describing the resolution. Here again, the merchant-side piece of the FV proxy intercepts the mail (Step 7), and must in this case **UpdateStatus** on the **PCR** (Step C) with the final resolution of the transaction, either

**PaymentComplete** or **Failed**. After each status update at the **PCR**, the new information is passed to the **Monitor** objects (Step D) which take application specific behavior, possibly ignoring the **InProgress** updates, or informing the user. If the payment status is complete, then the **PCR** sends the information held in its **Receipts** field.

It is important to note that everything above the dotted line in Figure 5 is independent of the particular payment mechanism. In Figure 4, the application needed to know how to form e-mail messages to First Virtual. With the abstraction of an **AccountHandle** and **PCR**, however, (as we will see in the next section) a different payment mechanism could be substituted below the dotted line with no disruption to the application. This flexibility is the goal of U-PAI.

## 6 Sample Ecash Proxy

Developed by David Chaum of DigiCash, ecash is an electronic "coin"-based payment mechanism which provides anonymity for the purchaser. Although the technical details are complex [4], they are not directly of concern to U-PAI, which interacts with ecash at the level of the user operations. For this discussion, we assume the text-based interface to the system used in the cyberbucks ecash trial. The steps in an ecash payment are enumerated below, and shown graphically in Figure 6.

1. The payer initiates the payment by entering a command either in the ecash process or directly at the UNIX shell. The command specifies an amount, a destination host and port, and a reference string.
2. The ecash software withdraws an appropriate number of coins from the user's account to make the payment, and transmits them to an approved bank for verification.
3. Assuming the coins are legitimate and have not been spent, the payee (merchant) is asked to approve the deposit.
4. The payee approves the deposit, sending a message to the ecash bank.
5. The bank sends the coins to the merchant, for deposit into the merchant's core account.
6. The payee application queries the payee account to determine whether the coins have arrived, using an ecash command entered directly from the UNIX shell or via the ecash interface.

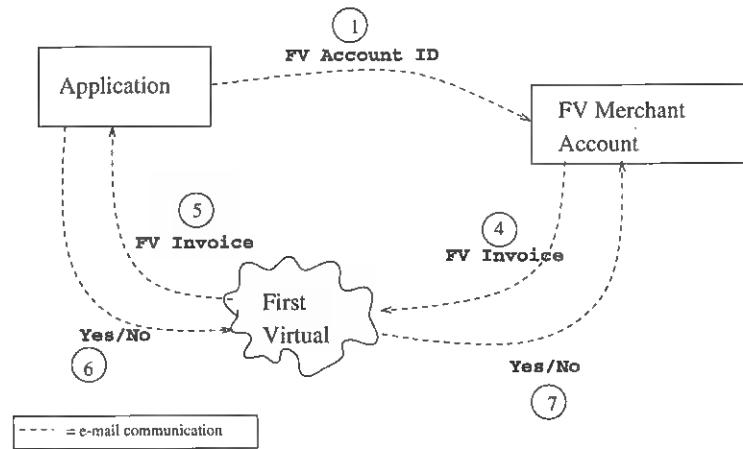


Figure 4: The steps involved in a First Virtual payment.

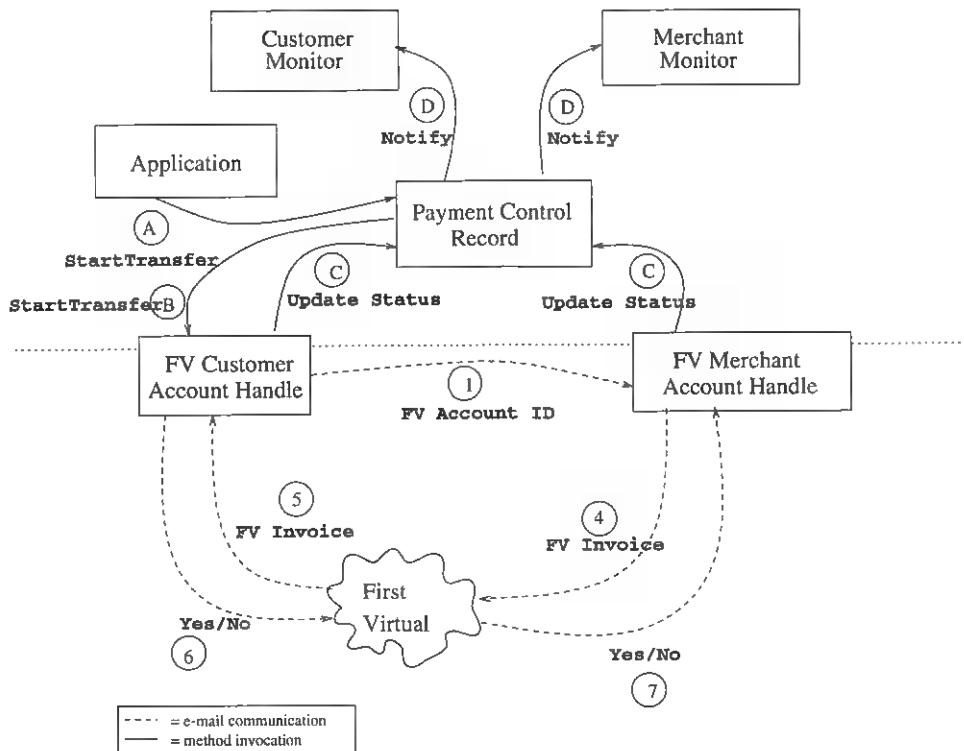


Figure 5: The steps involved in a First Virtual payment used with U-PAI.

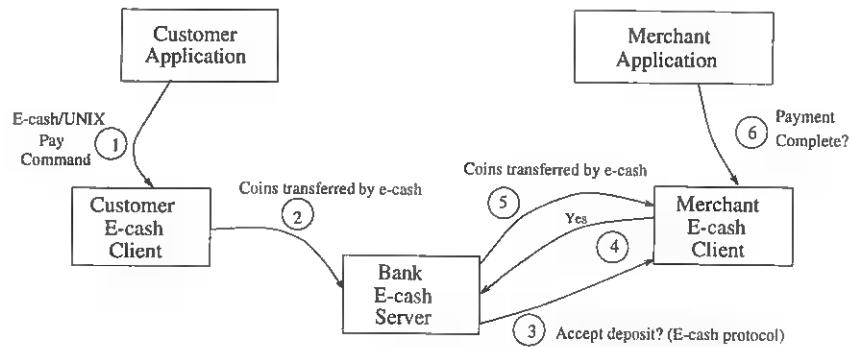


Figure 6: The steps involved in an ecash payment.

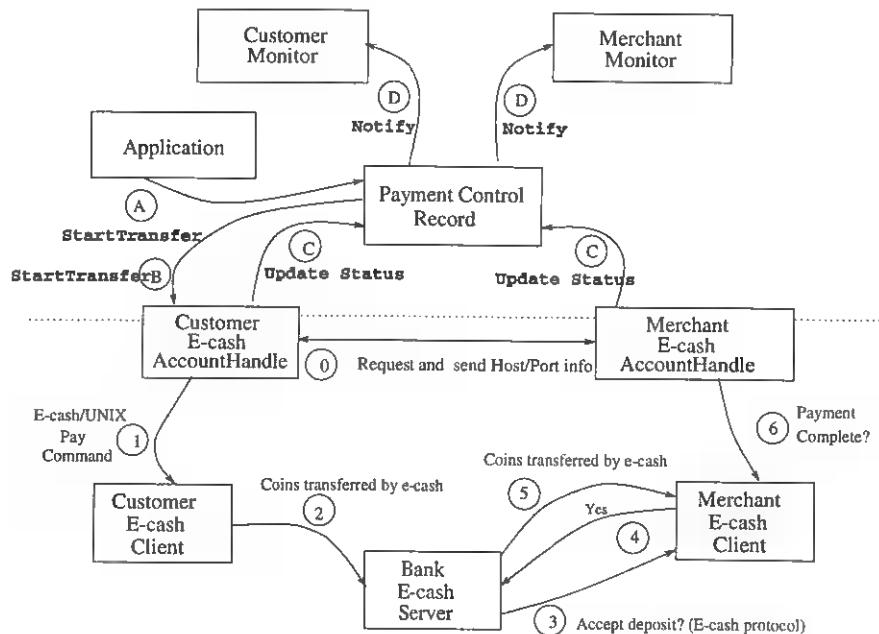


Figure 7: The steps involved in an ecash payment used with U-PAI.

The **AccountHandles** act as the proxy to the payment mechanism, as with the First Virtual system. When the **StartTransfer** is invoked, the destination **AccountHandle** is available to the source **AccountHandle**. The source **AccountHandle** calls a mechanism-specific method (not part of the definition of U-PAI) defined on ecash **AccountHandles** to learn the host address and port of the destination account (Fig. 7, Step 0). With this information, the source **AccountHandle** formats and executes an ecash pay command (Step 1). At this point, the ecash module takes over, contacts the bank, and verifies the coins (Step 2). The request for approval which the ecash bank sends to the payee (Step 3) is intercepted by the proxy on the merchant side, and automatically approved in Step 4 (if automatic approval is not acceptable on all payments, the source **AccountHandle** can notify the destination **AccountHandle** of the coming payment). The coins are then transferred to the merchant (Step 5), again through ecash specific code. The merchant's ecash **AccountHandle** determines when the payment is complete (Step 6) and triggers an update to the PCR (Step C). The status update from the PCR is sent to the monitors on the **MonitorList** (Step D), allowing the applications to be informed of the final disposition of the ecash payment, using the same status values (**PaymentComplete**, **Failed**, or **InProgress**) from the First Virtual proxies. If the status is **PaymentComplete**, then the PCR distributes the information recorded in its **Receipts** field. The interaction of the ecash system with the U-PAI interface is shown in Figure 7. Again, notice that the machinery above the dotted line is identical to that in Figure 5.

## 7 Failed Transactions and Security

In this section, we consider the behavior of the system in a few selected failure modes, such as network disturbances or frozen accounts. In some cases, the system design allows completion of a commercial transaction even under adverse circumstances. For instance, in the event that the ecash bank server is down, the **StartTransfer** method will recognize its inability to contact the bank, and notify the listening **Monitor** objects, perhaps enabling the buyer to select a different payment mechanism which is currently operable. Similarly, an ecash charge for which there are insufficient funds will result in an error condition being sent to the **Monitor** object, enabling alternative arrangements to be made.

The system is not foolproof, however. If a user initiates a payment using First Virtual and then receives no update because the e-mail was delayed, the user is uncertain of what to do. The status may show only **InProgress** with no indication of what step is currently ongoing, or how much longer is required before the process will be resolved. This ambiguity highlights one of the design decisions of U-PAI. In an effort to promote ease of implementation, no guarantees are offered about the completion of transactions—the mechanism and system operate on the “best effort” principal. In particular, under certain failure conditions with certain payment mechanisms, it may be impossible for the payer to prove that the payee received payment. By providing fine-grained specification of the transaction status to the **Monitor** objects through the **MinorStatus** values, however, along with the power to abort a transfer, the system provides maximum flexibility to its users. If a payment mechanism provides the capability to query the status of a particular transaction, an additional level of recovery is possible, because a **Monitor** object can use the **AccountHandle**'s **GetStatus** method if the **PCR** fails.

Also, security is not explicitly discussed in this paper. For the CORBA-based U-PAI methods (above the dotted line in Figs. 5 and 7), we assume the presence of a mechanism which provides access control on a per-method, per-object basis. This may be implemented using access capabilities building on the **Authorization** fields of the PCR. Other mechanisms such as digital signatures may be substituted. The desired result is that certain objects are prevented from reading or modifying data fields or executing methods, while other objects are permitted partial or total access. For example, the **UpdateStatus** method on a PCR should only be called by **AccountHandles** involved in the transaction.

For those steps below the dotted line, we assume that the underlying payment mechanism handles security appropriately. Properties such as confidentiality and non-repudiation that are provided by the payment mechanism may require additional work to ensure they persist through U-PAI. The messages should also be encoded in such a way to resist eavesdroppers and replay attacks.

## 8 Conclusion

We have proposed a Universal Payment Application Interface, which allows a variety of payment mechanisms to be accessed by the same interface, easing the use of multiple payment mechanisms or the pro-

cess of switching between payment mechanisms. We have outlined how payment mechanism proxies (combination of modules on both user and merchant side) allow this API to be supported without modification of the underlying payment mechanism. Finally, we have provided a CORBA ISL file for programmers interested in supporting or using this interface.

## 9 Acknowledgments

The authors wish to thank Ali Bahreman for thought-provoking discussions and helpful suggestions. Also, Martin Röscheisen contributed to the design of InterPay[3], a preliminary version of this work. Finally, changes suggested by the referees improved the completeness and clarity of this work.

## References

- [1] Alireza Bahreman. Generic payment services: Framework and functional specification. Technical report, 1996. Proceedings of the Second USENIX Electronic Commerce Workshop.
- [2] Alireza Bahreman and Rajkumar Narayanaswamy. Payment method negotiation service: Framework and programming interface. Technical report, 1996. Proceedings of the Second USENIX Electronic Commerce Workshop.
- [3] Steve B. Cousins, Steven P. Ketchpel, Andreas Paepcke, Hector Garcia-Molina, Scott W. Hassan, and Martin Röscheisen. InterPay: Managing multiple payment mechanisms in digital libraries. In *DL '95 proceedings*, 1995.
- [4] DigiCash. DigiCash brochure. Available at <http://www.digicash.com/publish/digibro.html>, 1994.
- [5] Donald E. Eastlake, 3rd. Universal payment preamble. Technical report, CyberCash, Mar 1996.
- [6] Steven H. Low, Nicholas F. Maxemchuk, and Sanjoy Paul. Anonymous credit cards. In *Proceedings of the 2nd ACM Conference on Computer and Communication Security*, 1994.
- [7] Mark S. Manasse. The Millicent protocols for electronic commerce. In *Proceedings of the 1st USENIX workshop on Electronic Commerce*, 1995.
- [8] Andreas Paepcke, Steve B. Cousins, Hector Garcia-Molina, Scott W. Hassan, Steven P. Ketchpel, Martin Röscheisen, and Terry Winograd. Towards interoperability in digital libraries: Overview and selected highlights of the Stanford digital library project. *IEEE Computer Magazine*, 29(5), May 1996.
- [9] L.H. Stein, E.A. Stefferud, N.S. Borenstein, and M.T. Rose. The Green commerce model. Technical report, First Virtual Holdings Incorporated, October 1994. Available at <http://www.fv.com/tech/green-model.html>.

## 10 Appendix: CORBA Payment Mechanism ISL

```
INTERFACE UPAI (* Version 1.0. For current version see:  
                 http://www-diglib.stanford.edu/diglib/software/UPAI.isl *)  
IMPORTS  
    IAny, (* See: http://www-diglib.stanford.edu/diglib/software/IAny.isl *)  
    CosPropertyService  
        (* See: http://www-diglib.stanford.edu/diglib/software/CosProp.isl *)  
END;  
  
TYPE String = ilu.CString;  
  
TYPE Amount = RECORD  
    Number : REAL,  
    Units : String (* dollars, yen, etc *)  
END;  
  
TYPE RefIDType = String;  
  
TYPE AccountTypeID = String;  
  
TYPE AccountTypeIDList = SEQUENCE OF AccountTypeID;  
  
TYPE Monitor = OBJECT  
    METHODS  
        Notify(whom : PCR, status : StatusEntry)  
            (* Notify is called whenever the status of the transaction 'whom'  
             changes & this Monitor object was in the PCR. *)  
    END;  
  
TYPE MonitorList = SEQUENCE OF Monitor;  
  
TYPE AccountHandle = OBJECT  
    METHODS  
        CreateAccount(NewAccountInfo : CosPropertyService.PropertySet): IAny.Any,  
            (* Creates a new real-world account, with the appropriate identifying  
             information. Optionally returns an authentication token.*)  
        OpenAccount(AccountInfo : CosPropertyService.PropertySet): IAny.Any,  
            (* Creates a new electronic representation of the existing real-world  
             account with the appropriate identifying information.  
             Optionally returns an authentication token.*)  
        GetAccountType() : AccountTypeID,  
            (* returns the type of this account. *)  
        GetTransferAccountTypesFrom() : AccountTypeIDList,  
            (* returns a list of account types that this account can receive  
             money from. *)  
        GetTransferAccountTypesTo() : AccountTypeIDList,  
            (* returns a list of account types that this account can transfer to. *)  
        GetBalance() : Amount,  
            (* returns the amount of funds available for payment in this account. *)  
        GetCreditLimit() : Amount,  
            (* returns the credit limit for credit-based accounts. *)  
        GetMechanismProperties() : CosPropertyService.PropertySet,  
            (* returns the meta-data properties like cost, time, anonymity. *)  
        CloseAccount(),
```

```

(* close this account. No further transfers can be made. *)

DeleteAccount(),
(* close this account & eliminate the real world account, too*)

StartTransfer(p : PCR),
(* Called by the system, not application programmer, to start
the money transfer*)

TryToAbortTransfer(p : PCR),
(* Called by the system, not application programmer, to try to abort
the money transfer*)

GetStatus(RefID : RefIDType) : PaymentStatus
(* Returns the current status of the payment identified by RefID. *)

END;

TYPE MajorType = ENUMERATION
    PaymentComplete,          (* Money transferred from payer to payee*)
    InProgress,               (* Transfer started, not completed *)
    Failed                    (* Error in payment, see description field*)
END;

TYPE StatusEntry = RECORD
    MajorStatus : MajorType,
    MinorStatus : IAny.Any

(*Typical Values are strings:
    Aborted                  -- Payer requested abort
    NotSufficientFunds,       -- Not Sufficient Funds for payer
    UnauthorizedSourceAccount, -- Payer not authorized to make payments
                                -- from this account
    UnauthorizedDestAccount,  -- Payer not authorized to make deposits
                                -- to this account
    NonExistentDestinationAccount -- Payee account not recognized
    UnableToTransferToAccountType -- Payee account wrong type
    NoSourceAccountSelected    -- Neither open() nor create()
                                -- has been invoked on this handle
*)

    END;

TYPE PaymentStatus = SEQUENCE OF StatusEntry;

TYPE PCR = OBJECT

METHODS

    SetRefID(RefID : RefIDType),
    SetContextID(ConID : RefIDType),
    SetAmount(amt : Amount),
    SetMonitorList(Mlist : MonitorList),
    SetDestAccountHandle(dest : AccountHandle),
    SetDestAccountAuthorization(auth : IAny.Any),
    SetSourceAccountHandle(src : AccountHandle),
    SetSourceAccountAuthorization(auth : IAny.Any),
    SetReceipts(rcptlist : IAny.Any),

    GetRefID() : RefIDType,
    GetContextID() : RefIDType,
    GetAmount():Amount,
    GetMonitorList():MonitorList,
    GetDestAccountHandle():AccountHandle,

```

```
GetDestAccountAuthorization() : IAny.Any,
GetSourceAccountHandle() : AccountHandle,
GetSourceAccountAuthorization() : IAny.Any,
GetReceipts() : IAny.Any,

StartTransfer(),
(* Initiates the transfer described in the other fields of the
 data structure. Asynchronous, returning immediately, doesn't wait
 for funds to be transferred. *)

GetStatus() : PaymentStatus,
(* Returns the current status of this transaction. *)

TryToAbortTransfer(),
(* Attempts to abort the transfer of funds initiated
 for this PCR. There is no guarantee the abort
 will be successful. *)

UpdateStatus(stat : StatusEntry)
(* Called by payment specific level to report progress *)
END;
```



# Anonymous Atomic Transactions

Jean Camp  
*Sandia National Labs*  
*Livermore, CA 94551*  
*ljcamp@ca.sandia.gov*

Michael Harkavy  
*Carnegie Mellon Univ.*  
*Pittsburgh, PA 15213*  
*bif@cs.cmu.edu*

J. D. Tygar  
*Carnegie Mellon Univ.*  
*Pittsburgh, PA 15213*  
*tygar@cs.cmu.edu*

Bennet Yee  
*UC San Diego*  
*La Jolla, CA 92093*  
*bsy@cs.ucsd.edu*

## Abstract

We show here an example of a protocol that satisfies anonymity properties while providing strong ACID (atomic, consistent, isolated, durable) transactional properties, resolving an open question. This allows us to provide electronic commerce protocols that are robust even in the event of message loss and communication failures. We use blind signature tokens to control values. We use a separate transaction log to reduce trust assumptions between the merchant, the consumer, and the bank.

## 1 Introduction

Consumer privacy is an important goal of electronic payment systems. Some researchers have approached this question by adopting a token-based model. These tokens are meant to act as a type of currency: they can be used to purchase a good, but like coins, they do not reveal the identity of the holder. These systems offer privacy in making a purchase. Some typical examples of token-based electronic payment protocols (“digital cash” protocols) are [2, 3, 7, 5, 15]. These protocols provide consumers with the ability to make anonymous purchases, purchases which can not be tracked by a bank to identify the purchaser. A stronger form of anonymity can be considered — anonymity in which the identity of the purchaser is hidden from both the bank and the merchant selling the goods.

But what happens when things go wrong? If the network (or merchant server) goes down during a purchase, how can users complain about non-delivered goods? If their purchases are anonymous, how can they prove that they really did pay and did

---

This work was supported in part by the Defense Advanced Research Projects Agency (ARPA contract F33615-93-1-1330), the National Science Foundation (NSF cooperative agreement IR-9411299), the US Postal Service, and Visa International. This work is the opinion of the authors and does not necessarily represent the view of their employers, funding sponsors, or the US Government.

not receive the goods? How can electronic judges and merchants adjudicate these complaints? How can they determine whether the consumer was really denied the goods, or whether the consumer is just trying to illegitimately acquire merchandise for free? And how can a consumer obtain satisfaction when the purchase is anonymous? These questions are especially important because the Internet today is an unreliable network — anyone who has spent some time browsing the web knows that communications often fail. Unscrupulous consumers and merchants will certainly attempt to take every advantage of system failures.

To illustrate the problem, consider the following simplified digital cash protocol: consumers pay for electronic goods with tokens. These tokens are anonymous, but designed so that if the consumer ever uses the same token twice, the consumer’s identity is revealed. Suppose a consumer pays for a good, but before she can receive acknowledgment that the merchant received payment, the network fails. Now, what can the consumer do? She doesn’t know whether the merchant received the payment or not. She has two basic strategies:

- She can *spend* the token again, by returning her token to the bank or spending it with a second merchant. But then, if the first merchant really did receive the token, she may be creating a race condition. Whoever gets the token to the bank first will get the money. Worse, when both tokens do reach the bank, the consumer will be accused of double-spending. Now, one can imagine variations on the digital cash protocol where a consumer might file a special type of complaint with a bank, but the design of this variation is non-trivial. Most types of variations will either reveal the consumer’s identity, allow a new type of fraud, be subject to ambiguous results if a message is not delivered, or have other undesirable effects. This topic was addressed at length in [4, 17, 18].
- She can *wait* and not spend the money. But in

this case, the consumer has locked up her funds. If the merchant did not receive her payment, then the consumer may be waiting for a very long time!

A standard approach to addressing the question of reliability is the notion of ACID (atomic, consistent, isolated, durable) transactions [10]. In the distributed systems community, ACID transactions have been widely adopted as the standard mechanism for realizing distributed transactions. The payment transactions should be *failure-atomic*, so that failures in parts of the system will not leave the entire system in some ambiguous, intermediate state.

How can we interpret these transactions in the context of electronic commerce? Tygar [17] has proposed using the classification below. Tygar began by assuming a model where consumers are purchasing electronic goods and services that will be delivered over a network (such as WWW page, for example). For tangible physical goods, alternative definitions are required to properly satisfy the atomicity property (motivating a multi-billion dollar industry in tracked, receipted courier delivery of messages and packages!) Tygar defines three classes of atomicity for electronic goods.

- **Money atomic** transactions feature atomic transfer of electronic money — the transfer either completes entirely or not at all. In money atomic protocols, money is not created or destroyed by purchase transactions.
- **Goods atomic** transactions are money atomic and also ensure that the consumer will receive goods if and only if the merchant is paid. Goods atomic transactions provide an atomic swap of the electronic goods and funds — similar to the effect of “cash on delivery” parcels.
- **Certified delivery** protocols are goods atomic and also allow both the consumer and merchant to prove exactly what was delivered. If there is a dispute, this evidence can be shown to a judge to prove exactly what goods were delivered.

Using this classification, we can see that the simplified digital cash protocol described above is *not* money atomic. The obvious question is: are anonymous atomic transactions possible? [17] This has been an open question.

Our first attempt to solve this question would be to use standard techniques to make a digital cash transaction atomic. The standard method for doing this is *two-phase commitment* [1, 9, 10, 11, 12].

In short, in two-phase commitment, one party assumes the role of transaction coordinator. That party knows and records the identities of all other parties in a non-volatile log. Each of the parties records its state before the transaction begins. As the transaction moves forward, various parties complete their required computation. Before changing the permanent store of those values, the parties send a message to the coordinator indicating that they are ready to commit. (Alternatively, they may abort the transaction by sending a negative message to the coordinator.) After receiving ready messages from all parties, the coordinator issues a commit message to all parties, causing the transaction to become permanent. Alternatively, if the coordinator receives an abort request or if the coordinator can not establish contact with one of the parties, the coordinator can abort the transaction by sending an abort message; in that case, all parties reverse the computation that they conducted towards the transaction.

So, as we see, the two-phase commit protocol requires that at least one party participating in the protocol (the transaction coordinator) knows the identity of all the parties involved. Additionally, two-phase commit assumes a *fail-stop* fault model, where the parties to the protocol can fail by stopping due to a crash, but not by lying or otherwise trying to cheat. In electronic commerce protocols, of course, we must be able to tolerate arbitrary Byzantine faults; one way to do this is to provide sufficient auditing information to detect these faults and later assign responsibility. This makes the standard two-phase commit protocol inappropriate for use in anonymous electronic commerce systems.

An alternative approach to this problem was attempted by Jakobsson [14], where the payment protocol is divided into two halves. Here, the digital cash is “rip-spent”: after the first half of the spending protocol, the consumer has committed to buying from the merchant but has not yet spent the money — some partial information is transferred, so that if the consumer attempts to abort the transaction, the digital cash is either lost (becomes unusable), or the identity of the consumer is revealed. This approach, unfortunately, is not satisfactory: each of the half protocols themselves may be interrupted, leaving the digital cash again in an ambiguous state.

## 1.1 Our contribution

Let us return to the open question mentioned above: Can anonymous transactions be atomic? Some researchers have speculated that the answer is negative. However, in this paper, we reverse this com-

monly held belief by answering the question affirmatively. We present a set of protocols for electronic commerce transactions which combines anonymity and atomicity while requiring very limited trust assumptions. We prove the goods atomicity properties of our protocols. For certified delivery we provide two variations:

- **one-sided certified delivery** where the consumer can prove what goods were delivered in case adjudication is needed (e.g., the goods do not match their description). The merchant is also guaranteed to be paid if and only if the consumer successfully obtains the electronic goods. On the other hand, the merchant can not prove that the consumer successfully received the goods promised. (This is the protocol presented in Section 3.3. As we argue below, if the burden of proof is on the consumer, then this method suffices to allow the consumer to prove the results of the transaction.)
- **two-sided certified delivery** which provides proof of the delivery of specific contents to both parties.

As we discuss below, there are tradeoffs between these two properties — two-sided certified delivery still keeps the identity of the consumer anonymous, but it does reveal some information about a cryptographic key used by the consumer. (This normally is not a problem since a consumer is expected to choose unique cryptographic keys for each transaction. However, we do discuss reusable keys below in Section 8.1, and if this option is chosen, there is a shade of privacy lost. This variation is presented in Section 8.6.)

It is important to emphasize that we have not designed this protocol with performance considerations in mind. The efficiency of digital cash protocols is already controversial [16, 17] and our mechanisms also require large amounts of computation. The intellectual contribution of this paper lies in showing that the widely held belief that atomic transactions can never be anonymous is not correct.

Section 2 explains our basic cryptographic and system assumptions. Section 3 presents a high level overview of the design of our protocols, followed by a more detailed description of our protocols. Section 4 informally analyzes the correctness of the protocols, and gives an argument for verification of the one-sided certified delivery property. Section 5 describes the logging requirements for all the parties in our system. Section 6 details the privacy properties of our system, giving an exhaustive list of the

types of information available to the various parties. In Section 7, we review the basic trust assumptions required by our system and show that strengthening these assumptions leads to simplified versions of the system. Section 8 presents several variations to our basic protocols, allowing partial spending of the withdrawn amount, greater efficiency by permitting public key reuse, etc. Finally, Section 9 concludes the paper.

## 2 Assumptions

In designing our protocols, we made a variety of assumptions about the capabilities of the participants and made use of several cryptographic tools. This section describes our assumptions and introduces some of the important tools which will be used.

Four parties participate in our protocol: a consumer  $C$ , a merchant  $M$ , a bank  $B$ , and a transaction log  $L$ . Now, of course, our parties are designed to allow multiple, scalable, simultaneous transactions that do not interfere with each other — this is the *isolation* requirement in ACID transactions. Thus, there can be many more than four parties — but in a single transaction, there will only be four parties.

All parties can perform basic cryptographic operations, e.g. cryptographic hash computation, signature computation and verification. All parties have well-known or verifiable public keys. All signatures can be verified by the receiving party.

To justify the claim of anonymity, the identity of the consumer must be protected from all other parties (that is, we make assumptions of *strong anonymity*).

We assume that the communications channels (in particular those between the consumer and the other parties) are anonymous, i.e. no information about the identity of the consumer is gained by communicating with the consumer. Now, in practice, this may be a questionable assumption — can't a merchant or a third party infer the consumer's identity through the TCP/IP return address on his packets? Supporting this assumption in an implementation may involve a fifth party, an anonymizer, which the consumer trusts not to reveal identity information.[8]

To provide atomicity, a modified, cryptographic version of the two-phase commit protocol is implemented using an external, publicly accessible transaction log. The transaction log receives and records messages, and then reproduces the recorded messages. The log localizes the global commit decision

to a single entity. The log also acts as a time-keeper, determining when to abort transactions due to a time-out.

Communication channels are assumed to be secure. The method used to implement this security (e.g. public keys) is not important to the functioning of the protocol. Some messages could be left unsecured, improving efficiency at the cost of a limited loss of privacy, but the details of this are not currently addressed.

Chaum [6] made a critical discovery that enabled the idea of digital cash to take place: *blinded tokens*. The use of blinded tokens is essential to our protocols. A token is a piece of data which can be created only by a specific issuer. Creation of a token without assistance from the issuer should be computationally infeasible. Blinded tokens are created by an interaction of a consumer with the issuer (the bank). After the interaction, the consumer has knowledge of a token which the issuer can not specifically identify. That is, the issuer does not know which token (from the range of valid tokens) the consumer has obtained. The tokens used in this protocol will have the additional property that each token, denoted  $Q^*$ , specifies the public half (including modulus), denoted  $Q$ , of a public key pair.

It is assumed that the consumer has an account with the bank, and that the bank can mint blinded token currency. This requires the bank to maintain token information as detailed in Section 5.

The transaction protocol given assumes minimal preparation and delivery costs for the goods. Goods must be prepared and delivered, although not in a usable form, prior to any guarantee of payment to the merchant.

Message signatures are computed on hash values of the plaintext, and then appended to the plaintext to form a signed message. This is relevant in the first step of the purchase protocol, 1, for efficiency reasons. It is also relevant in the second step, 2, so that the bank can read  $Q^*$  in order to determine  $Q$ . This assumption can be dropped with minor changes to these steps.

### 3 Protocols

In this section, we first give a high level overview of our anonymous atomic transaction system design, and then describe our abstract protocols.

Critical to our system is the use of a blind signature in the withdrawal protocol. Here, the consumer obtains a blinded token from the bank as a result of withdrawing money from the consumer's account.

Unlike previous works where the blind-signed data is a token which represents value, in our protocol the public key of a newly generated public/private key pair is signed; this certifies a trapdoor function rather than data to be disclosed in the purchase protocol. This effectively provides a temporary, anonymous certificate of ownership of the withdrawn amount. The private key of the key pair is known only to the consumer, and it is used with the certificate to anonymously authorize transfers of the withdrawn amount to a merchant's account. Authorization messages signed with this key are used in our transactions to signal readiness to commit to a purchase transaction, and to serve as part of the "paper trail" to prove that the token has been expended.

In the purchase phase, the merchant delivers encrypted goods along with a signed contract providing the goods description and the price. If the consumer finds the contract acceptable, readiness to commit is sent to the bank in the form of a signed message (using the above blind-certified key) to authorize the transfer of funds if the transaction commits, and then the bank similarly signals its readiness to the merchant with a message promising an anonymous deposit into the merchant account when the transaction commits. The transaction commits when the transaction log records a message from the merchant which contains the merchandise key.

Timely delivery of the merchant's message to the transaction log results in the transaction committing, thereby crediting the merchant's account and releasing the merchandise key to the consumer. If the merchant's message does not arrive before the expiration time, the transaction aborts.

Note that unlike standard two-phase commit, there is no central transaction coordinator; instead, the various parties' readiness to commit are determined using non-repudiable messages in a distributed, cascading fashion as explained in section 4.

Next, we give a detailed description of the withdrawal and purchase protocols.

#### 3.1 Notation

We use the following notation to describe steps in a protocol.

1.  $X \rightarrow Y \text{ messagetext} — \text{label}$

Here, the step number of the message is given (this is the first message in the protocol), the message is sent from  $X$  to  $Y$ , the text of the message is *messagetext*, and the step is named *label*.

We use the notation  $(message)_p$  to indicate the *message* is signed with public key  $p$ , and  $E_k(field)$  to indicate that a field is encrypted with symmetric key  $k$ .

### 3.2 Withdrawal and Exchange

The consumer generates a public key pair to use with each withdrawal. The public half of the pair is used to form *blinded-request*. The steps of the this protocol are given in Figure 1. At the end of the protocol, the consumer can form the token  $Q^*$  by unblinding *signed-blinded-request*. The contents of  $Q^*$  specify the public key  $Q$  (including modulus), whose private half is known only to  $C$ .

A token may be anonymously exchanged for a new token in a similar fashion by replacing the consumer's public key with the token's single-use key. The token exchange steps are given in Figure 2.

The following is an example protocol using a specific blinding technique. The bank has an RSA public key pair with modulus  $N_t$ , public exponent 3, and private exponent  $t = 3^{-1} \bmod \phi(N_t)$ . The bank has also made public a cryptographic hash function  $h$ .

1.  $C$  generates a desired public key pair  $Q, q$  with modulus  $N_q$
2.  $C$  selects a random number  $r \bmod N_t$
3.  $1. C \rightarrow B \quad h(Q) \cdot r^3 \bmod N_t$
4.  $B$  computes  $(h(Q) \cdot r^3)^t \equiv h(Q)^t \cdot r \bmod N_t$
5.  $2. B \rightarrow C \quad h(Q)^t \cdot r \bmod N_t$
6.  $C$  computes  $r^{-1} \bmod N_t$  and then  $h(Q)^t \bmod N_t$
7.  $C$  has  $Q^* = Q, (h(Q)^t \bmod N_t)$

The bank may use multiple signature keys for its blind signature, corresponding to different *brands* of tokens. The brand of a token determines its denomination and its withdrawal date. Having token brands is important for limiting the data logging requirements for the bank: until a brand of blinded token is withdrawn from use, the bank must maintain a database containing auditing information proving that expended tokens have been spent in order to prevent double spending (see Section 5). By *a priori* declaring that tokens will be worthless after the brand withdrawal date, the bank limits its data logging obligations; furthermore, brand withdrawal will also limit risk, since it limits the amount of time attackers will have to attack the key. Next, we discuss how the blind-signed token obtained above is used in the purchase protocol.

### 3.3 Purchase

Some negotiation of the transaction contract is assumed to take place prior to the transaction steps listed below. Given the current approach, the method of this negotiation has no direct bearing on the protocol. As above, each message is annotated with a mnemonic which describes the purpose of the message, and which will be used to refer to the message. For example, *authorization* denotes the authorization action by the party identified in the subscript. The protocol is an example of linear commitment in the sense defined in [10]. The steps of the purchase protocol are given in Figure 3.

In step 1, the merchant sends a signed copy of the contract and goods to the consumer. It is essential that the contract (*contract*) contain a description of the goods in order to provide one-sided certified delivery. The goods (*goods*) are encrypted with a single-use private key ( $k$ ), referred to as the merchandise key. The merchandise key will be revealed to the log and the consumer if the transaction commits. The message includes a transaction number ( $n$ ) generated by the merchant which should be different from any previously generated transaction number from the same merchant. A duplicate number could be detrimental only to the merchant. Upon receipt, the consumer verifies that the contract is acceptable.

In step 2, the consumer decides upon an expiration time (*expiration*) for the transaction, after which the transaction is considered to have failed, and the token can be reused or replaced. The consumer also selects a transaction log ( $L$ ) for the transaction. Both the bank and the merchant have effective veto power over the consumer's selection of  $L$  and *expiration*, since they will not provide authorization before knowing these values. The consumer then signals to the bank its readiness to commit by specifying the transaction and the token and key to be used ( $Q^*$ ). The bank must verify the validity of  $Q^*$ , including a check against reuse. The bank then uses  $Q^*$  to check the message's signature.

In step 3, the bank tells the merchant that it and the consumer are ready to commit, and includes in the message the value of the token (*value*) to be used for payment. The merchant verifies the transaction number is correct and that the token value, log identity, and expiration time are acceptable.

In step 4, the merchant commits to the transaction by sending the merchandise key to the log, along with the time-out, transaction number, and a signature. Upon receipt, the log verifies that the expiration time (*expiration*) has not passed.

W1.  $C \rightarrow B$  (*blinded-request*)<sub>c</sub> — withdraw<sub>c</sub>  
 W2.  $B \rightarrow C$  signed-blinded-request — withdraw<sub>b</sub>

Figure 1: Token withdrawal protocol.

E1.  $C \rightarrow B$   $Q^*, (\text{blinded-request})_q$  — withdraw<sub>c</sub>  
 E2.  $B \rightarrow C$  signed-blinded-request — withdraw<sub>b</sub>

Figure 2: Token exchange protocol.

In step 1, the log records the merchant’s commitment if it was received before *expiration*. The precise method of distribution of the recorded message is not important to determining atomicity properties, but some method for providing timely, good-faith delivery to the consumer is of practical importance. Any party can use this log record in conjunction with other signed messages obtained during the transaction to force the bank to transfer funds or to obtain the goods decryption key, completing the transaction.

In step 2, which may occur only after the *expiration*, the log generates a negative authorization at the request of the consumer or the bank. This indicates that no 4 for the given merchant and transaction number was received prior to the given expiration time. This allows the token to be freed for reuse or exchanged following a failed transaction; after *abort* is generated, the purchase protocol terminates.

## 4 Correctness and Atomicity

The atomicity properties of the protocol rest on the atomicity of the transaction log’s non-repudiable *commit* (or *abort*). The transaction log will eventually produce exactly one of *commit* or *abort* for any transaction. The other parties can use this, along with other data gathered during the course of the transaction, to prove that the transaction did (or did not) complete. Conversely, if a party claims that the transaction did (or did not) complete, it must be able to provide this proof to justify its claim to other parties.

The transaction protocol follows the two-phase commit model, however, the authorization actions of the parties are cascaded. First, the consumer authorizes the bank to lock the token to the transaction. Next, the bank authorizes the merchant to transmit

the key to the log. Then, the merchant sends the merchandise key to the log, authorizing the log to commit the transaction. Finally, the log issues the global commit. Accountability is similarly cascaded so that a party can be held accountable exactly when it has made an authorization and the transaction commits. For example, if the bank authorized the merchant to deposit the key (*authorization*<sub>b</sub>) without having received the consumer’s authorization to lock the token (*authorization*<sub>q</sub>), then the bank could be held accountable by the merchant (who would have *commit* and *authorization*<sub>b</sub>), but the consumer could not be held accountable by the bank (which would have *commit* but not *authorization*<sub>q</sub>).

The merchant can use *commit* and *authorization*<sub>b</sub> to prove that the bank should credit the merchant’s account with the value of the token. This provides the bank with *commit*. Possession of *commit* assures the bank that it will not be subject to a claim that the transaction has failed, since such a claim would require *abort*.

The bank can use *commit* and *authorization*<sub>q</sub> to justify (to the consumer) marking the token spent and denying reuse or replacement. The consumer can demand this proof, requiring the bank to produce *commit*, which contains the merchandise encryption key, thus giving the consumer access to the goods. Note that in practice, assuming good faith, the consumer will acquire the key prior to this, and demand of proof will not be necessary.

The consumer can use *abort* and *q* to demand that the token be unlocked. This provides the bank with *abort*. Possession of *abort* assures the bank that it will not be subject to a claim that the transaction has completed, since such a claim would require *commit*.

Finally, the consumer can use *commit* and *goods*<sub>m</sub> to prove the contents of the delivered goods. The goods encryption key, the encrypted goods, and the

- P1.  $M \rightarrow C$  ( $n, contract, E_k(goods)_m$ ) —  $goods_m$
- P2.  $C \rightarrow B$  ( $n, expiration, M, L, Q^*$ )<sub>q</sub> —  $authorization_q$
- P3.  $B \rightarrow M$  ( $n, expiration, M, L, value$ )<sub>b</sub> —  $authorization_b$
- P4.  $M \rightarrow L$  ( $n, expiration, k$ )<sub>m</sub> —  $authorization_m$  or
- P5. 1.  $L$  (( $n, expiration, k$ )<sub>m</sub>)<sub>l</sub> —  $commit$   
 2.  $L$  (( $n, expiration, M$ , failed)<sub>l</sub>) —  $abort$

Figure 3: Purchase protocol.

description of goods (contained in the contract) have all been signed by  $M$ , and *commit* proves that the transaction completed. Some means for review of goods by an outside authority should be available to establish claims of incorrect or fraudulent goods delivery.

From a correctness standpoint, it is important to examine the use of combinations of signed, non-repudiable messages employed above. Any given *commit* (or *abort*) is valid for only one combination of  $n$ , *expiration*,  $M$ , and  $L$ ; it is considered to be compatible only with the messages which agree on those values. The one exception is that the certificate  $goods_m$  does not mention  $L$  or *expiration*, and thus must agree only on  $n$  and  $M$  to be compatible. At each step, a party provides a signed message which can be used to prove that party's accountability with exactly the same range of *commit* messages as it would use in proofs of the preceding party's accountability. Thus, if a party is held accountable for the transaction, then it is provided with the non-repudiable messages that it needs to hold the previous party accountable as well.

## 5 Data Management

Our protocols rely heavily on the ability of the participants to hold each other accountable by maintaining records of each other's non-repudiable messages. We now discuss the record keeping required of the different participants in the protocol.

The consumer stores all data and messages received on all active tokens or transactions. This includes the token ( $Q^*$ ), the signed contract and goods ( $goods_m$ ), and finally the global commit (*commit*). The consumer can use  $goods_m$  and *commit* to prove the contents of the goods and the contract. If the goods are satisfactory, then the consumer may discard all data on the transaction.

The merchant stores the bank's authorization ( $authorization_b$ ) and then the global commit (*commit*). These are used to prove a completed transaction to the bank, which then issues a credit to the merchant. Once the bank has issued the credit, the merchant may discard all data on the transaction.

The transaction log stores the merchant's authorization ( $authorization_m$ ) whenever it produces a global commit (*commit*). This information may be discarded after some delay following the transaction expiration (*expiration*). The delay should be long enough that denial of access to the log for that duration is extremely improbable. It is also important that the log not generate *abort* for any transaction with an *expiration* which has been exceeded by more than this delay period.

The bank must maintain a variety of transaction information to correctly manage the protocol. The bank must have a selection of *brands* of tokens. The brand of a token specifies the method used to create the token as well as the properties (e.g. denomination) associated with the token. Different brands of tokens are used to cover the range of desired token properties, particularly denominations and expiration dates. Since the bank knows the brand of the blinded token withdrawn by a consumer, the denominations and expiration dates should have a coarse granularity so that many tokens of each brand are issued. The bank must maintain a database of processing information for each brand of tokens it issues.

The database for a brand of tokens tracks the status of tokens of that brand. During the transaction phase, the bank receives an authorization ( $authorization_q$ ) to commit a token to a transaction. This message is logged in the database, so that attempts at token reuse can be detected. Once the bank receives the the commit (*commit*) or abort (*abort*) for a transaction, it stores that as well. If the transaction completes, the bank should keep both

*authorization<sub>q</sub>* and *commit* until some period following the expiration for the brand of token used. If the transaction aborts, the bank need keep only *abort*. In order to limit the time the bank must store *abort* records, transaction claims by the merchant should have a limited time of validity, perhaps some fixed period following *expiration*. This period should be sufficiently long to allow time for reasonable delays or for outside party conflict resolution. If the period is based on *expiration*, then the bank may decide to not authorize any transactions with excessively late *expiration* values.

## 6 Privacy

An important consideration in any transaction is what information is revealed about the participants and to whom. In this section, with the aid of a table, we detail what information is obtained by various agents.

Table 1 gives the types of information available to various parties in the style of [4]. The entries for the merchant, the consumer, the bank, and the transaction log are based on their original information plus any information received over the course of a transaction. The information for law enforcement with warrant assumes record-keeping on the part of the bank, and law enforcement's knowledge of the item is dependent on merchant records. The electronic observer's knowledge is based upon performing traffic analysis on the encrypted messages. In the basic protocol, the transaction log is publicly readable, and thus an observer can also obtain full information about the merchant's identity.

## 7 Trust

In this section we discuss the assumptions of trust necessary for our protocols. We then consider two modifications based on alternative trust assumptions.

While there are many places where a dishonest participant or saboteur could delay progress or prevent commitment (e.g. by disrupting a communication channel), there is only one location where a corrupt coalition may benefit illegitimately. For this reason, there is one trust assumption required by the protocol; the merchant must trust the log to record received messages. If the log, in collusion with the consumer, fails to produce *commit*, but simply passes *k* (which is contained in *authorization<sub>m</sub>*) to the consumer, then the consumer will gain access to the goods while the merchant will not have *commit*,

and thus will not be able to demand payment. In practice, if the time to expiration is sufficiently long and the log is accountable for responsiveness, this sort of fraud might be detected. Trusted outside observers could notice that the log is failing to respond in reasonable time and take some action.

This trust assumption against a log-consumer coalition is a reason for the existence of the transaction log as a separate entity. Before he commits to the transaction, the merchant knows the identity of the log, and therefore he need only commit if the specified log is trusted. In practice, the selection of the log might be decided in the initial negotiation between the consumer and merchant. If the merchant is assumed to trust the bank not to conspire with the consumer, the transaction protocol can be simplified by merging the bank and the log.

The second reason for a separate transaction log is the consumer's desire for timely access to the goods. From a practical standpoint, the consumer must trust that the log will not intentionally delay passing the key to the consumer. Although the key must eventually be revealed to the consumer for the bank to justify crediting the transaction, this would likely take place on a much larger time scale than would be desirable for key delivery. If the log is required to satisfy some responsiveness guarantees, then limited delays can be enforced with the assistance of a trusted outside party. If the consumer is assumed to trust the merchant to make timely delivery of the key (given that it must be delivered eventually), then the transaction protocol can be simplified by merging the merchant and the log.

## 8 Protocol Variations

The protocols presented in this paper form the groundwork for many variations which alter or extend their functionality. In this section we describe modifications to support key reuse, multiple token transactions, partial token spending, cryptographic time-stamps, a non-public transaction log, and full certified delivery.

### 8.1 Reusable Customer Keys

One variant of these protocols permits reuse of *Q* at the expense of allowing the bank to link repeated uses of *Q*: instead of blind signing *h(Q)*, the bank blind signs *h(Q, s)*, where *s* is some serial number chosen by the consumer. In this fashion, many tokens ( $Q^* = Q, s, h(Q, s)^t$ ) can all specify the same key (*Q*). Only the bank sees *Q* (in 4), and so only

Information Party	Merchant	Consumer	Date	Amount	Item
Merchant	Full	None	Full	Full	Full
Consumer	Full	Full	Full	Full	Full
Bank	Full	None	Full	Full	None
Transaction Log	Full	None	Full	None	None
Law Enforcement w/warrant	Full	None	Full	Full	Full
Electronic Observer	Partial	None	Full	Partial	None

Table 1: Information Available with Anonymous Certified Delivery

the bank can link repeated uses of the same key to each other.

## 8.2 Multiple Token Transactions

To pay for items of arbitrary values, we may need to combine several tokens in a single transaction. In this case 2 must contain the various tokens, and be signed with all keys associated with those tokens. Let  $\hat{Q}^*$  be a list of tokens (possibly of different brands)  $(Q_1, h(Q_1)^{t_1}), \dots, (Q_m, h(Q_m)^{t_m})$  and let  $\hat{q}$  be the list of corresponding private halves  $q_1, \dots, q_m$ . We extend the subscript notation to vectors to indicate signing the plaintext with each private key in the vector. The new 1 step would then be

1.  $C \rightarrow B \ (n, \text{expiration}, M, L, \hat{Q}^*)_{\hat{q}}$   
authorization<sub>q</sub>

Multiple token transactions combine well with the use of one key for many tokens as discussed above, since this might reduce the number of signatures needed.

## 8.3 Partial Token Spending

The protocols may also be changed to support spending tokens in a check-like fashion. By including a particular value in 1, a consumer can authorize that only a part of a token's value is spent. The remaining value of the token may be exchanged for new tokens, or may be used in further purchases until all the value is used. Partial spending of tokens is compatible with both key reuse and multiple token transactions.

## 8.4 Cryptographic Time-Stamps

An important function of the log is to time-stamp the arrival of 4. The time-stamps used should

include clock time information, since transaction authorization expiration will be in terms of real time. In order to reduce the trust that the parties must place in the log's honesty, cryptographic time-stamping [13] may also be employed. Cryptographic time-stamping will give the additional property that if the log is compromised, the log entries made prior to the time of compromise may still be trusted.

## 8.5 Encrypted Log Entries

In order to facilitate anonymous key acquisition by the consumer, the transaction log is publicly readable. While the logged message (*commit*) does not contain sensitive information, it might be used to determine the merchant's identity. Extra privacy could be supported by including a secret key (*s*) in the purchase messages. In fact, if *n* is required to be randomly selected and is sufficiently large, then *n* could be used as this secret key. The logged message would be encrypted using the secret key so that only the parties of the transaction could read (*commit*). To support efficient lookups, a function on known data could be used to generate indices for log entries (e.g.  $E_s(M)$ ).

For even greater privacy, the log could be left unaware of the secret key and simply time-stamp, sign, and record any received messages (and their indices). This would require a modification of the *abort* message to indicate that no message with the given index was available at a specified time. Additionally, *expiration* should be left in plaintext so that the log can know not to publish messages with timestamps greater than their *expiration* values.

## 8.6 Two-Sided Certified Delivery

The last and most intricate variation on the protocols is the addition of support for two-way certified

- CD1.  $M \rightarrow C$  ( $n, contract, E_k(goods)_m — goods_m$
- CD2.  $C \rightarrow M$  [ $goods_m]_q — goods_q$
- CD3.  $C \rightarrow B$  ( $n, expiration, M, L, Q^*)_q — authorization_q$
- CD4.  $B \rightarrow M$  ( $n, expiration, M, L, Q, value)_b — authorization_b$
- CD5.  $M \rightarrow L$  ( $n, expiration, Q, k)_m — authorization_m$  or
- CD6.CD1.  $L$  (( $n, expiration, Q, k)_m)_l — commit$
- CD2.  $L$  (( $n, expiration, M, failed)_l — abort$

Figure 4: Full Certified Delivery purchase protocol.

delivery, which is detailed in Figure 4. Our protocols provide one-sided certified delivery; only the consumer can prove what goods were delivered. If the burden of proof is expected to fall on the merchant, then the purchase protocol can be changed to provide full certified delivery at the cost of extra complexity. First, we introduce the notation  $[M]_x$  to indicate the signature of  $M$  with key  $x$  without the plaintext, e.g.,  $h(M)^t \bmod N_t$  for RSA signatures. If we provide the merchant with  $Q$  and  $[goods_m]_q$ , then the merchant will be able to prove what goods were delivered to the holder of  $Q$ . The merchant must additionally be able to prove that the holder of  $Q$  is the consumer for whom the transaction was processed. Our purchase protocol for certified delivery follows:

The new step, 2, supplies the merchant with the signature by  $q$  of the goods description. The inclusion of  $Q$  in 3 enables the merchant to link  $Q$  with the payment to be received. The logging of  $Q$  in CD1 associates  $Q$  with the completed transaction. To increase the trustworthiness of this association in case of corruption by one or more parties, the cryptographic time-stamping variation described above should be employed. In variants where  $Q$  may be reused, the log entries should be encrypted to prevent unassociated parties from linking the repeated uses of  $Q$ .

## 9 Conclusion

In this paper, we presented protocols for achieving anonymous atomic transactions, answering an open question[17].

As stated in the introduction, these protocols are not being proposed for use in their current form. Both efficiency concerns and legal concerns — portions of the protocol may violate financial institution recordkeeping requirements on transactions

over \$100 stipulated by the Money Laundering Act (12 USC §1829) in the US — must be addressed before such a protocol can be used. But an existence proof of an anonymous atomic protocol is an important step towards providing reliable, secure electronic commerce on the Internet, while maintaining individual privacy. Our variant protocol designs demonstrate the range of anonymous, ACID transaction available.

We hope that researchers and system designers in the electronic commerce community will further explore the technical feasibility of providing anonymous atomic electronic money transactions in real systems. We believe that these are fascinating technical issues and that in some contexts anonymous and reliable transactions will have important social value.

## References

- [1] Andrea J. Borr. Transaction monitoring in Encompass: Reliable distributed transaction processing. In *Proceedings of the Very Large Database Conference*, pages 155–165, September 1981.
- [2] Stefan Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, Centrum voor Wiskunde en Informatica, 1993.
- [3] E. Brickell, P. Gemmell, and D. Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 457–466, 1995.
- [4] L. Jean Camp, Marvin Sirbu, and J. D. Tygar. Token and notational money in electronic

commerce. In *Proceedings of the First USENIX Workshop in Electronic Commerce*, pages 1–12, July 1995.

[5] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology — CRYPTO '88 Proceedings*, pages 200–212. Springer-Verlag, 1990.

[6] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Crypto '82 Proceedings*, pages 188–293. Plenum Press, 1983.

[7] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[8] Benjamin Cox. Maintaining privacy in electronic transactions. Technical Report TR 1994-9, Carnegie Mellon University Information Networking Institute, Pittsburgh, PA, September 1994.

[9] C. J. Date. *An Introduction to Database Systems Volume 2*. Addison-Wesley, Reading, MA, 1983.

[10] J. Gray and A. Reuter. *Transaction Processing*. Morgan-Kauffman, 1993.

[11] James N. Gray. A transaction model. Technical Report RJ2895, IBM Research Laboratory, San Jose, California, August 1980.

[12] James N. Gray. The transaction concept: Virtues and limitations. In *Proceedings of the Very Large Database Conference*, pages 144–154, September 1981.

[13] Haber and Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2), 1991.

[14] M. Jakobsson. Ripping coins for a fair exchange. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology: Eurocrypt 95 Proceedings*, volume 921 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[15] Steven Low, Nicholas F. Maxemchuk, and Sanjoy Paul. Anonymous credit cards. Technical report, AT&T Bell Laboratories, 1993. Submitted to *IEEE Symposium on Security and Privacy*, 1993.

[16] Bruce Schneier. *Applied Cryptography, 2nd edition*. John Wiley and Sons, 1996.

[17] J. D. Tygar. Atomicity in electronic commerce. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 8–26, May 1996.

[18] Bennet S. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.



# Strongboxes for Electronic Commerce

## (Extended Abstract)

Thomas Hardjono<sup>1</sup> and Jennifer Seberry

Centre for Computer Security Research  
University of Wollongong  
Wollongong, NSW 2522  
Australia  
email: thomas/jennie@cs.uow.edu.au

## Abstract

As electronic commerce becomes a reality, additional services related to electronic commerce will also emerge. The current paper proposes the provision of *electronic strongboxes* as an integrated part of the wider electronic commerce. The strongbox concept is introduced as an electronic counterpart of physical strongboxes typically found in large traditional financial institutions, such as banks, having secure vaults or other secure physical storage. The work identifies some requirements of electronic strongboxes both from the functionality perspective and from the security point of view. A simple framework for an electronic strongbox system is also presented.

*Keywords:* Electronic Strongboxes, Electronic Commerce, Payment Systems, Distributed Systems.

## 1 Introduction

Electronic commerce on the Internet has become one of the major issues in computing in the last few years. The development of the world-wide-web technology and its related browsers has transformed the idea of commerce and trading on electronic media into a reality. The vast opportunities presented by electronic commerce can be easily gauged by the amount of serious interest shown by the business sector and by researchers in the field of computing.

In the business sector the number of Internet Service

Providers have increased dramatically, responding to the ever greater number of people wishing to "connect to the net". The term *network computer* has been coined to capture the duality of the nature of personal computers today, namely as a desktop computer and as a gateway to the world of the Internet.

From the computer research sector quite a number proposals have been put forward for the immediate use of the Internet as a payment media through which users can carry-out transactions and payments linked to the existing physical financial infrastructure (eg. DigiCash [1, 2], iKP [3], NetBill [4] and SET [5] to name a few). Other schemes suggest the use of electronic cash or coins to be used as a circulating currency on the Internet (eg. Net-Cash/NetCheque [6, 7]).

As electronic commerce becomes a major activity on the Internet (and other interconnected networks), users will demand other related services to be delivered through and by the Internet. We perceive that one such service will be the provision of *electronic strongboxes* as a counterpart to the existing physical strongboxes, typically found in large financial institutions.

In the traditional financial sector the provision of strongboxes has been in service for sometime. Customers can apply to have a private strongbox held within a bank, in which the customer can place any type and any amount of valuables, subject only to the physical characteristics of the strongbox. The bank typically has no interest in the contents of the strongbox, and it derives income from providing safe storage and access to such strongboxes. The identity of the strongbox customer and the fact of the customer having a strongbox are usually treated as

<sup>1</sup>The author is also at the University of Western Sydney - Macarthur, NSW 2560, Australia.

confidential by the bank.

In this paper we carry-over the notion of strongboxes from the physical reality into the digital world. We propose the introduction of electronic strongboxes on the Internet as an integrated part of the electronic commerce infrastructure. It is our belief that electronic strongboxes can play an important role for the safe keeping of important items (in their electronic representation). We identify some functional requirements of electronic strongboxes which bear some resemblance to that of electronic payment protocols (Section 3 and Section 4). In addition, we present a simple framework for an electronic strongbox system, describing some of the basic interactions among the participants of the system (Section 5). The security requirements of this system is briefly discussed in Section 6. In Section 7 some security technology considerations related to the implementation of electronic strongboxes are discussed. Some remarks and conclusions in Section 8 end the paper.

## 2 Lockers along the Super-Highway

In today's banking world the provision of strongboxes for customers is a common occurrence. Customers typically place important items, such as jewelry and important documents, in strongboxes. Access to the strongboxes is dependent on the bank that provides the strongbox service. Usually, a bank would require a customer to identify himself or herself before access is provided. Ideally, however, access should be provided to any person when that person reveals a key that is recognized by the bank. Hence, *anonymity* of the carrier of the key is guaranteed. With the physical strongbox storage, two general approaches are usually provided:

- Customer access to a private environment containing all the non-removable strongboxes (ie. drawers).
- Customer access only to their own removable strongbox, within a private environment (ie. actual boxes)

With the current interest among the business community in conducting trade on the Internet, the notion of an electronic version of strongboxes is an interesting and attractive one. In the electronic world, and within the context of electronic commerce, banks and other certified organizations would

provide the electronic strongbox service to their customers on the Internet. Customers may apply for such a strongbox over the Internet, and payment for the service can be done using electronic cash or other electronic payment forms. The customer would then have access to their respective strongboxes over the Internet, using secure browsers which allow them to place electronic items in their strongbox. There is almost no limit to the variety of electronic items that can be stored in an electronic strongbox. Some of the typical items may include:

- Electronic coins or cash
- Electronic bank cheques
- Digital documents (eg. contracts)
- Anonymous digital certificates of ownership of physical items
- Cryptographic material to access other services

A customer may have multiple strongboxes, each at differing institutions along the electronic "super-highway". Access can be provided for 24-hours per day, and the customer would be able to move items among her or his own collection of strongboxes, or two customers can exchange items that will be stored in their respective strongboxes.

Similar to the physical world, in the electronic world access can be delegated by a customer to another person by way of the customer giving the access key (or its suitable derivative) to that person. In such circumstances, the person carrying the key can access the corresponding strongbox while remaining anonymous to the institution.

A third party maybe appointed for such cases when disputes occur between an owner of a strongbox and the institution that maintains the strongbox. This may occur, for example, when a dishonest user claims that his or her access key has a matching strongbox within the bank, or when the bank inappropriately denies access to a valid owner of strongbox.

Other institutions may act as *valuers* and *converters* of legal physical items where valuable items (eg. gold) are given a valuation and an electronic certificate for the item is generated. The same institution may also provide long-term safe storage for the physical items, whilst the anonymous owner uses the electronic certificate on the Internet. The certificate can then be used for personal trade or *Barter*, which is something common in everyday life. In this context,

strongboxes can play an important role in facilitating those non-monetary commerce in an untraceable manner. Legal items may also be advertised anonymously as being “For Sale” over the Internet, with the valuers and other trusted third parties being the point of contact. In effect, these strongboxes can become a type of secure “public storage” media, where individuals can disperse their electronic properties all over the Internet, with the storage management and the actual location of their physical data being transparent to the user.

The concept of anonymous storage itself is not new. The early work by Brandt *et al* [8] points to the benefits of anonymous and verifiable databases, particularly in the context of privacy against government bodies that wish to cross-correlate data belonging to individuals in society. In [8] the true identity of each individual remains unknown and the individual employed a different *pseudonym* [9] when dealing with each government body or institution. The main feature of the work was that each individual must also have the ability to verify that his or her personal details held by an institution are correct. Further related work has also been reported in [10].

However, one underlying difference between the anonymous/verifiable database concept and the public strongbox concept is the privacy of the data. In the anonymous/verifiable database, it is intended that the institution that maintains the database view the data belonging to the users, whilst at the same time maintaining the anonymity of the users. The users can verify that the database contains correct data about the user. A typical example would be a hospital database containing sensitive medical data belonging to patients. A patient may have personal details that are important for medical requirements (eg. blood type, diabetes, etc), but at the same time the anonymity of the patient must be upheld to prevent discrimination against some patients with serious illness (eg. cancer, HIV, etc) that could affect their entitlements (eg. health insurance, medical benefits, etc) and affect their social standing. In contrast, in the electronic strongbox concept the contents of the strongbox must remain private, with the users still remaining anonymous and being able to verify and modify (insert/remove) the contents of his or her strongbox at anytime.

In general, electronic items stored in a strongbox should be enciphered individually by its owner before they are placed in the strongbox. This approach would then allow strongbox access to be implemented in two ways:

- The owner is given the entire strongbox which she or he must open using the key, after which he or she may obtain individual items (which must be deciphered).
- The owner “delegates” the institution to open his or her strongbox and to deliver to the owner specific (encrypted) items. The institution remains unable to view the specific item requested, as the items are enciphered by the owner.

## 3 Functional Requirements

There are a number of basic requirements which must be fulfilled by electronic strongboxes, following the requirements of their physical counterpart. These are listed and briefly discussed in the following.

### 3.1 Anonymity

#### A1 Anonymity of owner.

The owner must always remain anonymous, and the fact that she or he owns a strongbox must also remain a private fact. Methods to create pseudonyms exist in other forms of electronic commerce which can be used in the strongbox case.

#### A2 Anonymity of key holder.

The key holder is the user that presents a valid key to the bank to access a strongbox held by the bank. The bank has the right to verify that the key fits into one of its strongboxes, and to deny access if the verification fails. The key holder can be the owner of the strongbox or any other user delegated to access the strongbox by its owner.

### 3.2 Privacy

#### P1 Privacy of strongbox contents.

As in the case of physical strongboxes, the contents of the strongbox should remain undisclosed to all parties except the key holder opening it using a valid key. Any system implementing the strongbox should ensure that the institution providing the service does not have backdoor or other hidden channels to access or view the contents of the electronic strongbox.

In addition, the strongbox should be tamper-resistant from the institution itself, who might attempt to illegally remove or add items to the strongbox. This may be achieved using cryptographic techniques (eg. hashing, signing) to provide the owner with proof and assurance that the strongbox has not been tampered with since it was last accessed.

In the physical world, some level of trust exists between the bank and strongbox owner, whereby the owner relies on the bank not to place hidden cameras designed to view the strongbox contents and that the bank will not tamper with the strongbox. Ideally, such trust should also exist between a customer and the strongbox provider, similar to the level of trust between merchant and acquirer [3, 5].

**P2** *Privacy of strongbox locations.*

A user may have multiple strongboxes scattered all over the Internet under different guarding institutions. The locations of these strongboxes should be private information, available only to the owner (or any other delegated user) and the respective institutions. One institution should not be aware that its customer also owns strongboxes elsewhere.

**P3** *Access to strongbox only by a key holder.*

The institution must without exception provide access to the strongbox only to the key holder that presents a valid key.

A security mechanism must be employed to provide at least two levels of verification, namely at the point of request for access to the strongbox, and later at the point of the opening strongboxes. These two levels can be implemented cryptographically, and should eliminate possibilities of procedural errors.

**P4** *Storage of a variety of electronic items.*

An electronic strongbox should be able to store a variety of digital items, subject only to the agreed storage space limitations. Even such limitations should be easily and immediately negotiable when a user reaches his or her storage limit.

System parameters that protect the strongboxes must be maintained under secure and tamper-free storage at the institution.

### 3.3 Contents Transferability

**C1** *Items exchangeable between strongboxes.*

Analogous to the physical counterpart, electronic strongboxes must allow for the exchange of items between two (or more) strongboxes. Strongboxes may belong to the same owner, or they may belong to different owners who are working together.

**C2** *Untraceability of moved items.*

Since the contents of strongboxes must remain private, moved items must then be untraceable. Untraceability should hold regardless of how many times an item has been moved between strongboxes, and regardless whether or not the item finds its way into a strongbox within which it previously resided. That is, a strongbox should not have a “memory” of its previous contents.

### 3.4 Delegations

**D1** *Strongbox key can be delegated.*

Similar to the physical strongboxes, any person carrying the appropriate key must be able to open the electronic strongbox. Ideally strongboxes should even allow stolen keys to be used, as the issue of protecting keys is separate from user anonymity.

In the banking sector some banks do provide the owner with some protection against stolen keys. However, methods that require user identification can also result in the user’s identity being revealed.

In electronic strongboxes, delegation should be provided, whereby an owner of the strongbox can delegate another user to become a key holder in order to access the owner’s strongbox. Both users must remain anonymous. At the same time, delegation schemes must have a limited lifetime or the ability to be revoked by the owner [11].

Single-use keys may provide a solution, in which delegated keys are derived from the original key, and where the bank holding the strongbox are aware of a key being a derivative, and would allow only one-off access to a given strongbox. Multiple-use keys may also be devised, using technology similar to electronic coins. Every usage of the key would reduce its worthiness, until it is diminished when it reaches its maximum number of usages.

## 4 Additional Functional Requirements

The flexibility of the digital world presents a number of opportunities to provide features of electronic strongboxes which are infeasible or difficult to achieve in the physical world.

### 4.1 Movable strongboxes

Electronic strongboxes should be be movable between institutions, similar to the way electronic cash or coins are movable around the Internet. An owner of a strongbox must be able either to move the entire strongbox without opening it, or to shift the contents of one strongbox at one institution to another strongbox under a different institution. Both alternatives are attractive, and both should be available to the user, depending on the user's circumstances. Security, privacy and anonymity must be ensured in both cases.

### 4.2 Notification to the owner

Although the owner of a strongbox must remain anonymous through the use of pseudonyms, they must be available for notifications via their pseudonym. Notifications may include:

- Notification (confirmation) that the owner's strongbox was accessed at a particular date and time (or a failed attempt was made to access the strongbox).
- Notification of fees that are due to be paid by an owner of a strongbox

### 4.3 Electronic charges

Owners of strongboxes pay their fees using electronic currency. This can be done on a periodic basis, or long-term payments can be made upon the commencement of the strongbox.

When an owner is absent from the Internet for a long period or when the owner fails to answer notifications concerning overdue fees, the institution may take the strongbox off-line and keep it on secondary storage (eg. dump into CD-ROM). When the owner in future requests access to the owner's strongbox, the institution can bring the strongbox on-line only

after the due fees are paid by the owner. Proving ownership can be through the access key in the usual manner. Disputes with regards to payments must be resolved through a third party acceptable to the owner and the institution.

### 4.4 Designation of heir

An owner of a strongbox should be able to designate another valid pseudonym as an heir to be notified and given access in the case that the owner dies or the strongbox is never accessed over a long period of time (eg. years). This should occur if no prior arrangement was made by the owner with the institution regarding very infrequent access, and if the owner fails to respond to the various notifications about fees that are due. Other procedures must also be applied in the case that the designated heir fails to respond. In all these cases, the presence of a third party such as a lawyer or notary would be required as in the usual case.

The issue of property inheritance in the electronic world remains an interesting open problem, both from the legal aspect and from the economic aspect (eg. taxation in certain countries) [12].

## 5 A Simple Framework for a Strongbox System

In this section we propose a simple framework for a strongbox system (Figure 1), using components (ie. participants) typically found in electronic commerce systems. All electronic interactions between participants are assumed to be over a secure channel, with peer authentication conducted at the commencement of communications. The current proposal does not pretend to be comprehensive, and it attempts to address the main components only. Additional components will be required to support the framework to achieve full workability.

The participants of the system are as follows:

- *Customer*: the customer or user, interacting with the Strongbox Provider (eg. Bank) for the safekeeping of electronic items.
- *Strongbox Provider*: an institution that provides the electronic strongbox service to a customer.
- *Valuer*: the on-line Valuer is trusted to verify that an electronic item belonging to an owner

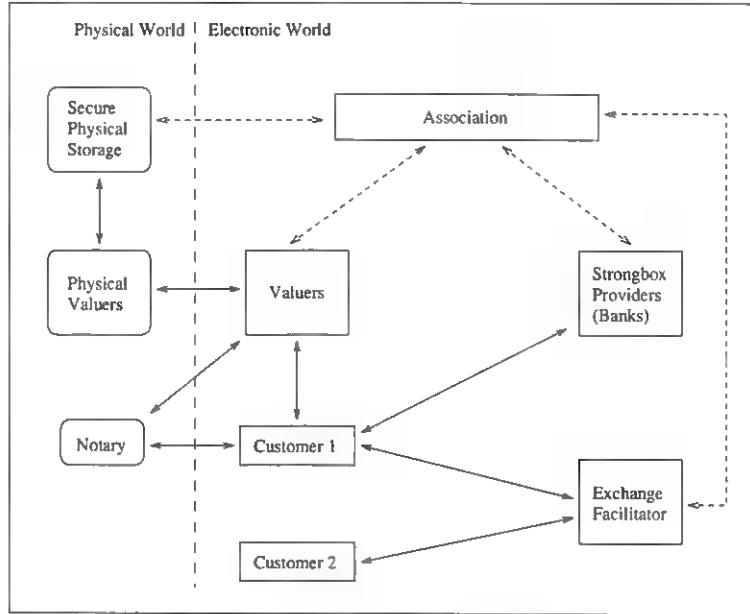


Figure 1: Electronic strongbox system

(ie. Customer) truly exists and has not been modified by its current owner. The Valuer can also be requested to split items into several sub-items, and issue certificates for them. Several Valuers may exist on-line, and each must recognize the other's certification.

- **Exchange Facilitator:** the Exchange Facilitator aids two or more Customers who wish to exchange items from their strongboxes. The Facilitator can be a Strongbox Provider and is under the jurisdiction of the Association.
- **Association:** the Strongbox Providers and the Valuer work under the umbrella of the Association. Customers bring disputes to the Association.

In addition, there are the *Physical Valuer* and the *Notary* which are in the physical world and interfaced to the electronic world. The Physical Valuer should be distinct from the on-line Valuer as the Physical Valuer knows what a physical item constitutes and which pseudonym forwarded the physical item to be valued. The Physical Valuer stores the physical items at the *Secure Physical Storage*, to which the Association has access in the case of disputes. The Notary comes in on behalf of a Customer when disputes necessitates their presence. In the remainder of this paper, unless otherwise stated, the term "Valuer" will refer to the on-line Valuer (as opposed to the Physical Valuer).

The Customer is the owner of the contents of a strongbox and is deemed also as the owner of the strongbox. The Customer obtains membership into the system through the Association which issues the Customer with the credentials (eg. within a smartcard) and with a pseudonym to be used within the system. The Customer henceforth employs this pseudonym when using the system. The Association may in fact be that which exists in the electronic commerce infrastructure and which oversees the usual electronic trading, purchases and payment. The Customer joins the strongbox service by opening an account with the Strongbox Provider, which can be a Bank or other institutions having the necessary computer infrastructure to provide this service.

In order to bring an item into the system the Customer must first obtain a valuation of the physical item to the Physical Valuer. The Physical Valuer issues the Customer with a digital certificate corresponding to the physical item. This certificate is recognized and accepted by all participants in the system. If requested, the Physical Valuer may attach a monetary value to each item, which may then be described in the certificate. The actual physical item itself is kept in the Secure Physical Storage, under the control of the Physical Valuer and/or the Association. Any Customer presenting an electronic certificate for a physical item can obtain the item from the Physical Valuer or through the Association.

The unit of the physical item to be valued and certified must be agreed upon between the Customer and the Physical Valuer (eg. six bars of gold can be written under one certificate, or six certificates can be produced corresponding to the six physical items). Having small units for the valuation allows for easier usage of the items at a later date. However, should a Customer wish to break-up an electronic item into several reasonable components – bearing in mind the physical reality of the item – the Customer can approach the on-line Valuer to obtain such services.

Here, although ideally any physical item should be allowed to be introduced into circulation, social/economic stability and order demands that illegal items (eg. drugs) be prevented from entering the electronic system. This prevention can be conducted at the Physical Valuer interface.

It remains an interesting and open problem as to whether a Customer must prove ownership of legal items. The extent to which electronic strongboxes should mimic physical strongboxes, particularly in the case of stolen goods, must be decided by authorities implementing the system.

Once within the system the certificate is referred to as an electronic item. What the item constitutes and who holds the item presently must remain confidential. A Customer can store the electronic item with any Strongbox Provider, assuming he or she already has a strongbox account with them.

When two or more Customers have agreed to exchange items, they can carry-out the exchange of the corresponding electronic items through the Exchange Facilitator. Ideally, before an exchange occurs, the Customers should prove the possession of the items to each other (eg. via zero-knowledge protocols). However, even without such pre-exchange confirmation of possession, the Exchange Facilitator must be able to ensure that no cheating occurs. The Facilitator must inform each Customer as to the electronic items it has received for the exchange instance (to prevent cheating), and the Facilitator must also provide a guarantee of non-repudiation should one (or both) Customer dispute the exchange. The Facilitator can be a trusted third party, or it can be one of the Strongbox Providers selected by both Customers.

The use of the Exchange Facilitator is optional. Customers can perform any exchange of items directly among themselves, through a secure channel. However, without the Exchange Facilitator disputes cannot be resolved and the burden of risks lie fully with

the Customers.

## 6 Security Requirements for the Strongbox System

Similar to electronic payment systems, a number of security requirements exist for the strongbox system to be reliable and workable. It goes without saying that the *authentication* of participants holds an important place before any interaction can occur. The *impossibility of forging* of electronic items must be guaranteed throughout the system. Finally, the requirement of *undeniability of actions* (or non-repudiation) carried-out by participants in the system. Some of the other more specific requirements are briefly presented in the following.

### Strongbox Provider Requirements

- *Proof of the retrieval of a strongbox.* The Provider must have some form of proof that a strongbox is currently being “checked-out”. That is, that the strongbox has been retrieved and is currently in the possession of the Customer. This is to prevent the Customer from claiming otherwise and therefore forcing the Provider to take account of losses. This notion is similar to that of the forging of electronic cash or coins, or to that of denying that payments have or have not been made.

The retrieve and store operations must exhibit the typical transaction properties of atomicity, consistency, isolation and durability [13, 14].

A further aspect that must be taken into consideration is the allowable length of time for a strongbox to be held (checked-out) by its owner and the implications on security. Given that a Customer typically knows the contents of his or her strongbox – either from human memory or through a list stored securely (eg. within a smartcard) – it is reasonable to assume that the check-out and check-in should occur within the span of a single transaction. A reasonable timespan would similar to that in which a merchant expects immediate payment from a purchaser.

- *Verification of access key of the strongbox.* Before providing a key holder with access to the claimed strongbox, the Provider must have sufficient proof that the requester (ie. owner

or their delegate) is a valid party within the system. That is, the requester has a valid pseudonym and can be authenticated. The Provider must also verify that the key is a recognized and valid key.

One potential problem would be the possibility of the illegal duplication of access information. That is, the potential that more than one access key exists at any time. Current technology can solve this problem either through smartcard systems or through the provision of a single-use access keys for the strongboxes. In the later case, a new access key needs to be generated each time a strongbox is retrieved and stored.

### Customer Requirements

- *Unauthorized retrieval of strongbox is impossible.* A Customer must have the assurance that the unauthorized checking-out of his or her strongbox is impossible. Stolen electronic items should be prevented from circulating without being detected.

Depending on the implementation, the certificate corresponding to the item may carry the pseudonym of its current owner (see the next Section). Since the certificate is unforgeably signed by the Valuer, stolen electronic items may be detected later at an Exchange Facilitator, a Valuer or at a Physical Valuer. A possible safe-guard can be implemented at the physical end, when Customers convert their electronic items back into physical items currently being stored in the secure physical storage.

- *Proof of storage by the Provider.* A Customer requires some proof in the form of a *receipt* that his or her strongbox has been correctly checked-in and that the Provider now holds the strongbox.
- *Proof of valuation.* When an electronic item undergoes valuation or when it is split by the Valuer into several electronic sub-items, a Customer owning the item (and thus the sub-items) requires proof that the Valuer currently holds the item, and also proof that the valuation has been carried-out. Clearly the Valuer itself must be a certified one and be authenticated by the Customer before any valuation transactions occur.
- *Proof of exchange transaction.* When a Customer carries-out an exchange of items with an-

other Customer via the Exchange Facilitator, both Customers must have sufficient proof that the exchange occurred correctly in such a way that neither party can deny the transaction.

### Valuer Requirements

- *Proof that valued item and valuation result has been received by Customer.* This is to prevent a Customer accusing the on-line Valuer of stealing an item submitted for valuation.

### Exchange Facilitator Requirements

- *Proof of exchange transaction.* Corresponding to the proofs required by a Customer for the exchange of an item, the Facilitator requires proof of the submission of the items to be exchanged, and more importantly proof of the delivery and receipt of the items after the exchange. This proof must come from all involved Customers, and serves as protection for the Facilitator against false claims by the Customers.

## 7 Technological Issues for Strongboxes

There are a vast number of issues related to the concept of electronic strongboxes and their implementation. It is beyond the scope of this introductory paper to cover each of them. Some of these, however, are briefly discussed in the following.

### 7.1 Representation of Electronic Items

There are many ways to represent items electronically. One possible method would be to employ two types of certificates for each item:

- *Item Certificate:* this is the electronic item itself in the shape of an unforgeable certificate and having a one-to-one correspondence with the physical item. The Item Certificate carries the signature of the Physical Valuer and is co-signed by an on-line Valuer.
- *Description Certificate:* this is a certificate guaranteeing that a given item exists somewhere in the system. The certificate may con-

tain a digest or hash of the Item Certificate, and is signed by the on-line Valuer. The certificate may contain the pseudonym of the current owner.

The two certificates are inseparable and should be stored together in the strongboxes. The aim of having a Description Certificate is to allow one Customer to prove its ownership to another Customer before an exchange occurs. During an exchange, both certificates are handed-over as an item unit.

The concept is derived from the idea of certified photocopies of important documents (eg. passports) which are often required for government and legal purposes. Periodically the Description Certificate must be renewed by way of the Item Certificate being reconfirmed by the on-line Valuer.

Similar to electronic cash, some form of serial numbering may be applied to all electronic items system-wide, to prevent illegal copying of certified items by its current owner. This must be done with the precaution that the serial numbers do not become way to trace the movement of items [15].

Upon an exchange between two Customers the Exchange Facilitator may request an on-line Valuer to re-certify electronic items as belonging to their new owners respectively. For each electronic item, both the Item Certificate and the Description Certificate must be signed by the on-line Valuer. The Description Certificate will then contain the pseudonym of the new owner of the corresponding item.

Note that no identity information, such as the pseudonym, is mentioned anywhere within the Item Certificate. Thus, the current owner of the Item Certificate may at any time obtain the actual physical item by presenting the Item Certificate to the Physical Valuer. The Physical Valuer must then inform the on-line Valuer of the removal of the item (via its serial number) from circulation within the electronic world.

## 7.2 Security on the User's Side

Security – or the lack of it – is currently one of the main obstacles to achieving the full use of electronic commerce on the Internet. Large financial institutions such Banks have the necessary resources to establish a reasonable level of security for their computing systems. However, security on the user's side is lacking. The vision of millions of users on their workstations or Home PCs conducting elec-

tronic commerce or trade must first address the need of trusted computing technologies at the user's end.

There are a number of potential approaches that can be taken to provide security at the user's end:

- Tamper-resistant technology. Tamper-resistant boxes can be provided as part of the internal hardware for the typical PC or Network Computer (NC). Smartcards can then be used to load specific security parameters to such boxes. The challenge in the future lies in making these affordable.
- Access terminals. Institutions can provide access terminals in the manner of Automatic Teller Machines. Besides providing physical security, such terminals can be available at the institution's premises. Although practical, this approach somewhat defeats the convenience of conducting electronic commerce from the user's desktop.
- Probe software. Although a contentious issue, the notion of down-loadable self-executing software is an attractive one. Here, an institution or a trusted third party can provide auto-executables which can be down-loaded (eg. via a browser) and which can perform automatic remote scanning or probing of a user's workstation or PC/NC to evaluate its security. This notion can be extended to situations where the software reboots the workstation and loads a specific secure operating system for the workstation. After the session, the previous local operating system can be reloaded. How this concept and its implementation can be extended over wide networks – and how acceptable it will be to the user community from the privacy perspective – remains to be seen.

## 7.3 Multilevel Secure Strongboxes

The idea of multiple strongbox providers lends immediately to the notion that strongboxes can have differing levels of security and therefore cost of maintaining them. The frequency of access also plays an influence on the costs of the strongboxes. Users with less valuable items may choose cheaper and "weaker" strongboxes, while for their expensive items they may choose the strongest strongboxes from the variety of strongbox Providers. Each strongbox Provider may offer either a uniform

strongbox type or provide differing levels of strongboxes.

## 8 Remarks and Conclusion

Having proposed electronic strongboxes as part of the electronic commerce infrastructure, this paper has attempted to identify some the functional and security requirements of electronic strongboxes. This effort does not pretend to be comprehensive, as there are a number of issues that remain to be resolved in the wider context of electronic commerce, and also within the specific scope of electronic strongboxes. It also does not ignore the fact that difficulties exist in any design and implementation of the concept. We believe, however, that the concept should be tied closely to developments in electronic commerce and payment systems, as these areas will represent the infrastructure within which the electronic strongbox concept can be comfortably implemented. The current paper has presented a simple framework for an electronic strongbox system, taking the more familiar participants from electronic payment systems.

The provision of electronic strongboxes as a service should not be too far in the future, as the security technology to implement it has partly arrived accompanying electronic commerce. There are a variety of issues which must be addressed to realize strongboxes in the wider context of electronic commerce. Some of these issue include, but not limited to:

- Anonymity of Customers, while providing the various features of electronic strongboxes.
- Interfacing electronic strongboxes with physical strongboxes at the Secure Physical Storage component.
- Value of items versus strongbox costs.
- Key escrowing of strongbox-keys by governments in some countries.
- Legal status of strongboxes when the owners are foreign nationals.
- Transferability of strongboxes across national boundaries.
- Item exchanges over national boundaries and the type of Exchange Facilitators that will thus be needed.

- Effects of converting electronic items back into physical items when the new owners are foreign citizens and its legal implications.
- Valuer infrastructure required for an international strongbox system.

These issues will be the subject for continuing research as they are important for the economic viability and technical feasibility of the electronic strongbox concept.

### Acknowledgements

We thank the anonymous referees for their useful comments and advice.

## References

- [1] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," *Communications of the ACM*, vol. 28, no. 10, pp. 1030-1044, 1985.
- [2] D. Chaum, "Achieving electronic privacy," *Scientific American*, pp. 96-101, August 1992.
- [3] M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner, "iKP - a family of secure electronic payment protocols," in *Proceedings of the First USENIX Workshop on Electronic Commerce*, (New York), USENIX, 1995.
- [4] M. Sirbu and J. D. Tygar, "NetBill: An internet commerce system optimized for network-delivered services," *IEEE Personal Communications*, pp. 34-39, August 1995.
- [5] Visa and MasterCard, "Secure Electronic Transaction," 1995. <http://www.visa.com>.
- [6] B. C. Neuman and G. Medvinsky, "Requirements for network payment: The NetCheque perspective," in *Proceedings of IEEE Compcon'95*, (San Francisco), IEEE, 1995.
- [7] G. Medvinsky and B. C. Neuman, "NetCash: A design for practical electronic currency on the internet," in *Proceedings of the First ACM Conference on Computer and Communications Security*, ACM, November 1993.
- [8] J. Brandt, I. B. Damgard, and P. Landrock, "Anonymous and verifiable registration

in databases," in *Advances in Cryptology - Proceedings EUROCRYPT '88 (Lecture Notes in Computer Science No. 330)* (C. G. Gunther, ed.), pp. 167–176, Springer-Verlag, 1988.

- [9] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–88, 1981.
- [10] T. Hardjono and J. Seberry, "Applications of smartcards for anonymous and verifiable databases," *Computers & Security*, vol. 14, no. 5, pp. 465–472, 1995.
- [11] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson, "Authentication and delegation with smart-cards," Technical Report 67, Digital Systems Research Center, October 1990.
- [12] R. Kalakota and A. B. Whinston, *Frontiers of Electronic Commerce*. Addison-Wesley, 1996.
- [13] L. J. Camp, M. Sirbu, and J. D. Tygar, "Token and notational money in electronic commerce," in *Proceedings of the First USENIX Workshop on Electronic Commerce*, (New York), USENIX, 1995.
- [14] L. Tang, "Verifiable transaction atomicity for electronic payment protocols," in *Proceedings of 1996 IEEE ICDCS16 International Conference on Distributed Computing System*, IEEE, May 1996.
- [15] D. Chaum, "Privacy protected payments: Unconditional payer and/or payee untraceability," in *Smart Card 2000: The Future of IC Cards* (D. Chaum and I. Schaümller-Bichl, eds.), pp. 69–93, Amsterdam: North-Holland, 1989.



# Model Checking Electronic Commerce Protocols

Nevin Heintze

Bell Laboratories

Murray Hill, NJ 07974

nch@research.bell-labs.com

J. D. Tygar

Carnegie Mellon Univ.

Pittsburgh, PA 15213

tygar@cs.cmu.edu

Jeannette Wing

Carnegie Mellon Univ.

Pittsburgh, PA 15213

wing@cs.cmu.edu

H. Chi Wong

Carnegie Mellon Univ.

Pittsburgh, PA 15213

hcwong@cs.cmu.edu

## Abstract

The paper develops model checking techniques to examine NetBill and Digicash. We show how model checking can verify atomicity properties by analyzing simplified versions of these protocols that retain crucial security constraints. For our analysis we used the FDR model checker.

## 1 Atomicity Properties

Correctness is a prime concern for electronic commerce protocols. How can we show that a given protocol is safe for use? Here we show how to use model checking to test whether electronic commerce protocols satisfy some given atomicity properties.

For verifying properties of protocols, model checking is a dramatic improvement over doing hand proofs, because it is mechanizable; it is a dramatic improvement over using state-of-the-art theorem provers because it is automatic, fast, and requires no human interaction. Moreover, we found a number of problems in proposed electronic commerce protocols using model checking. Model checking allows us to focus on just those aspects of the protocol necessary to guarantee desired properties. In doing so, we can gain a better understanding of why the protocol works and often can identify places of optimizing it.

For this paper, we have chosen to check atomicity properties. [2] argue that these properties are central to electronic commerce protocols.

In an *atomic* protocol, an electronic purchase either

- aborts with no transfer of money and goods; or

- fully completes with money and goods exchanged.

Moreover, these atomic properties are preserved even if communications fail between some of the parties, because of failure of either a communications link or a node (including the parties participating in the protocol.)

Tygar [22] gave informal descriptions of three protocol properties that appear to be related to atomicity:

**money atomicity** Money should neither be created nor destroyed by electronic commerce protocols. For example, this protocol is not money atomic:

1. Consumer sends message to consumer's bank: *transfer \$value to merchant*;
2. Consumer's bank decrements consumer's balance by *\$value*;
3. Consumer's bank sends message to merchant's bank: *increase merchant's bank balance by \$value*;
4. Merchant's bank increments merchant's balance by *\$value*.

If Message 3 is not received, then the consumer's balance will have lost money without the merchant's bank having received the money. Effectively, money is destroyed.

**goods atomicity** A merchant should receive payment if and only if the consumer receives the goods. Goods atomicity is particularly relevant in the case of electronic goods (such as binary files) that are delivered over the network. For example this protocol is not goods atomic:

1. Consumer sends credit card number to merchant;
2. Merchant charges consumer's credit card;

---

This work was supported in part by Defense Advanced Research Project Agency (ARPA contract F33615-93-1-1330), the National Science Foundation (NSF cooperative agreement IR-9411299), and by the US Postal Service. This work is the opinion of the authors and does not necessarily represent the view of their employers, funding sponsors, or the US Government.

### 3. Merchant sends electronic goods to consumer

Suppose Message 3 is not received by the consumer; then she will not have received the goods for which she was charged.

**certified delivery** In the case of electronic goods, both the merchant and the consumer should be able to give non-repudiable proof of the contents of the delivered goods. (We do not consider certified delivery in this paper.)

In this paper, we discuss how to use model checking to determine whether money atomicity and goods atomicity hold of two classes of electronic commerce protocols: account-based (e.g., NetBill [5, 21]) and token-based (e.g., a simplified protocol inspired by Chaum's offline DigiCash protocol [3, 4]). We used the FDR model checker [15], though other model checkers could have been used.

## 2 Model Checking

Model checking is a technique that determines whether a property holds of a finite state machine. The expression of the property and the machine could be in different languages where the property is a logical formula and the machine is described as a set of states and a state transition function. Since the machine has a finite number of states, an exhaustive search (through a standard reachability algorithm) is done to check that the logical formula holds at every state. The model checker SMV uses this approach. Alternatively, as with FDR, the expression of the property and the machine could be in the same language. In this case, a test for language inclusion is done to determine whether the property (one set of traces) holds of the machine (the second set of traces). Model checking is completely automatic, and usually fast, at least in comparison to alternative techniques like theorem proving.

An additional benefit that model checking provides over other techniques such as theorem proving is that if the property being checked does not hold, a counterexample is produced. This feedback is an invaluable means of debugging. It also is a way to explore the design space, perhaps finding ways to optimize a design (e.g., by eliminating a message exchanged or an extra encryption step).

In our context of using FDR as our model checker, we describe the protocol that we wish to analyze

through a CSP [10] process, Imp. This is the "machine" we want to check. We describe the property, e.g., money atomicity, that we wish to check of the protocol as another CSP process, Spec. To determine whether the protocol satisfies the property we test whether Imp is a *refinement* (in the technical sense used in CSP) of Spec. Roughly speaking, Imp's set of traces should be a subset of Spec's set of traces; thus, the machine is a trace refinement of the property. In reality, for atomicity properties, we need the failure refinement, which extends the trace refinement to handle non-determinism.

Model checking is a demonstrated success in hardware verification. Researchers and industrialists have used checkers like SMV [17], Murphi [6], COSPAN [9] and SPIN [11] to find bugs in published circuit designs, floating point standards, and cache coherence protocols for multiprocessors. It has been adopted by the hardware community to complement the traditional validation method of hardware simulation.

Model checking has also recently gained the attention of the software community. Notably, Atlee and Gannon used SMV to check safety properties for event-driven systems; Jackson uses his model enumeration method in Nitpick to check Z-like specifications [12]; and Vaziri-Farahani and Wing used SMV to check cache coherence in distributed file systems [23].

In the security domain, Roscoe [19] used FDR to prove noninterference of a simple security high-low hierarchy (after Bell-LaPadula's model) and Lowe [14] recently used FDR to debug and prove the correctness of the Needham-Schroeder authentication protocols [18]. Our work is the first to use model checking for analyzing electronic commerce protocols and for checking atomicity properties<sup>1</sup>.

## 3 Two Case Studies

We investigate NetBill and a simplified digital cash protocol with respect to money atomicity and goods atomicity. Money atomicity is concerned with the conservation of money in the context of account balances and electronic coins. That is, electronic coins should not be arbitrarily created or destroyed, and

<sup>1</sup>Use of formal methods to demonstrate protocol security has attracted wide attention – we cannot survey all the results in a brief paper such as this. However, electronic commerce protocols have received less attention – two important papers on formal (non-model checking) methods for electronic commerce are [1, 13].

fund transfers and conversions between funds and coins should be consistent. In other words, if we have a system formed by a consumer  $C$  and a merchant  $M$  who have accounts at a bank, the sum given by the formula

$$\begin{aligned} & C\text{'s account balance} + C\text{'s coins} \\ & + M\text{'s account balance} + M\text{'s coins} \end{aligned} \quad (1)$$

should be conserved. In the context of electronic coins, another component of money atomicity is that rightful possession of a coin should entitle the owner to spend the coin or deposit it in an account. This “cash property” will play an important role in our analysis of a simplified digital cash protocol.

Goods atomicity is concerned with the integrity of a sale: we want to guarantee that goods are transferred exactly when money is transferred. A consumer only wants to pay for goods received. A merchant wants to be paid for goods delivered.

### Assumptions

Our analysis focuses on the atomicity aspects of protocols. We do not, for example, consider the cryptographic details of the protocol—in fact our modeling of a protocol completely hides these details. We also do not model multiple interleaved runs of a protocol (in which, for example, a single agent could participate as a consumer in one run and a merchant in another). Instead we consider a single run of the protocol with one consumer, one merchant, and one bank. We discuss these abstractions further in Section 4.

Perhaps the most important assumption we make is about the failure model used in our analysis. First, consider the bank. In the context of bank failure, few if any atomicity properties can be guaranteed. In practice, banks go to great lengths to ensure reliable, fail-safe service. We model this by assuming that the bank never fails. Next, consider communication with the bank. This may take place over some unreliable medium such as a telephone line or the Internet. However, as a last resort, anyone can physically go to the bank to deposit funds or present purchase orders. In effect, every agent has a fail-safe communication line with the bank.

Now consider agents other than the bank. We allow communication between non-bank agents to fail arbitrarily. However, we only allow limited failures at non-bank agents because arbitrary failure would compromise atomicity properties. For example, sup-

pose that a merchant receives an electronic coin in exchange for goods, and then immediately fails before depositing the coin at the bank. The coin is effectively lost and money atomicity fails. Note, however, that the only party to suffer was the party that failed; there is no loss to the consumer nor the bank.

Our failure model for agents, other than banks, will be based upon the notion of commitment points, as used in standard database transactions [7, 16, 8]. We assume that each agent (other than the bank) has a particular point in the protocol at which that agent commits. Before this point is reached, we allow an agent to abort the protocol freely. After the commitment point, we consider only failures in an agent if the failure can potentially affect the outcome of the protocol for another agent. In particular, we ignore failures that can affect only the agent’s own outcome. In Section 5 we outline a more comprehensive failure model that expands these ideas.

### 3.1 NetBill

NetBill [5, 21] is designed to support very low-cost transactions involving electronic goods. One central and distinguished claim of the NetBill protocol is that it satisfies goods atomicity, and this will be the focus of our analysis. In NetBill, all money-related activities are centralized at the bank and take the form of transfers between accounts; consequently, arguing money atomicity is straightforward.

Here, we use an abstracted and simplified version of the protocol that captures atomicity properties. For full details on the protocol see [5, 22].

### The Protocol

The consumer  $C$  starts the protocol (Figure 1) by sending the merchant  $M$  a goods request, to which  $M$  responds with the goods encrypted with a one-time key  $K$ . At step 3,  $C$  sends  $M$  an electronic payment order (EPO) signed with  $C$ ’s private key. This EPO constitutes a fund transfer authorization, and sending it to  $M$  marks  $C$ ’s commit point.  $M$  checks the validity of this EPO, endorses it, appends  $K$  to it, and sends it to the bank  $B$ . This is the point where  $M$  commits to the transaction. Including  $K$  with the endorsed EPO is central to ensuring goods atomicity. At step 5,  $B$  sends to  $M$  a receipt of the fund transfer (which includes  $K$ ). Then  $M$  forwards this message to  $C$ . In case  $M$  does not forward the message (either because of failure, bad management,

1.  $C \rightarrow M$  : *goods request*
2.  $M \rightarrow C$  : *goods, encrypted with a key K*
3.  $C \rightarrow M$  : *signed EPO (electronic payment order)*
4.  $M \rightarrow B$  : *endorsed signed EPO, signed K*
5.  $B \rightarrow M$  : *signed receipt (including K)*
6.  $M \rightarrow C$  : *signed receipt (including K)*

If C does not receive the signed receipt, C may contact B directly:

7.  $C \rightarrow B$  : *transaction inquiry*
8.  $B \rightarrow C$  : *signed receipt*

Figure 1: The simplified NetBill protocol.

or attempted fraud), C can go to the bank for a copy of this message, and hence obtain  $K^2$ .

### NetBill in FDR

To model NetBill, we view each agent as a finite state machine, and use CSP processes to encode them. A CSP process denotes a set of sequences of events, where an event represents a state transition of the state machine; states are implicit.

Figures 2, 3, and 4 present simplified versions of the consumer, the merchant, and the bank processes respectively. Note that  $\rightarrow$  is a form of sequential composition,  $\parallel$  and  $[\cdot]$  are choice operators (with  $\parallel$ , the choice is completely arbitrary, whereas  $[\cdot]$  gives preference to unblocked processes), STOP denotes process termination, ! and ? are the communication primitives (for example,  $\text{coutm}! \text{goodsReq}$  sends the message `goodsReq` on the channel `coutm`, and  $\text{cinm}?x$  receives a message from channel `cinm`, and binds the variable `x` to the message).

CSP uses a synchronous model of communication between processes. Since we are modeling a distributed system, we need to simulate asynchronous communication. For communication from agent a to agent b, we use two channels `aoutb` and `bina`, and

<sup>2</sup>What if a corrupt merchant sends a bogus  $K$ ? In the Netbill protocol, in step 4, the merchant sends a signed version of  $K$  to the bank. Previously, in step 2, the merchant has sent the goods (encrypted with  $K$ ) to the consumer, and the hash of those has been included in the EPO and signed by both the merchant and the consumer. Hence, after the fact, the fraud and responsible party can be detected and proven to other parties. This is treated at length in [5, 22]. Our analysis in this paper does not consider the impact of bogus keys.

a process that reads anything from the first channel and writes it to the second. This process can be easily modified to introduce communication failures as needed.

Processes ABORT, SUCCESS, ERROR, NO\_FUNDS, NO\_TRANSACTION, END and FAIL are mapped to the CSP STOP process. We use them to improve readability of the code.

### Money and Goods Atomicity for NetBill

Recall the money conservation property given by the sum in Formula 1 at the start of this section. Since we do not have electronic coins, and we have only one run of the protocol, this property is satisfied exactly when a debit is matched by a credit, or vice-versa. In CSP, this can be specified as:

```
SPEC1 = STOP ||  
((debitC -> creditM -> STOP) []  
 (creditM -> debitC -> STOP))
```

Note that the third component of the specification,  $\text{creditM} \rightarrow \text{debitC} \rightarrow \text{STOP}$ , could be omitted, since in our specification of NetBill a `debitC` always happens before `creditM`, if they ever happen. To check this specification using FDR, we first combine the consumer, merchant and the bank processes with an appropriate communication process, and then hide all of the irrelevant events (the Appendix gives the details of the communication process, process combination and event hiding); call the resulting system `SYSTEM1`. Finally, we check that `SYSTEM1` is a refinement of `SPEC1` with the FDR command line:

```

CONSUMER = ABORT || coutm !goodsReq -> GOODS_REQ_SENT

GOODS_REQ_SENT = ABORT || cinm ?x ->
    (if x==encryptedGoods then ENCRYPTED_GOODS_REC
     else ERROR)

ENCRYPTED_GOODS_REC = ABORT || coutm !epo -> EPO_SENT

EPO_SENT = (cinm ?x -> (if (x==paymentSlip) then SUCCESS
                           else if (x==noPayment) then NO_FUNDS
                           else ERROR)) []
    (timeoutEvent -> coutb !transactionEnquiry -> BANK_QUERIED)

BANK_QUERIED = cinb ?x -> (if (x==paymentSlip) then SUCCESS
                             else if (x==noPayment) then NO_FUNDS
                             else (if (x==noRecord) then NO_TRANSACTION
                                   else ERROR))

```

Figure 2: The consumer process.

```

MERCHANT = ABORT || (minc ?x -> (if x==goodsReq then GOODS_REQ_REC
                                    else ERROR))

GOODS_REQ_REC = ABORT || (moutc !encryptedGoods -> ENCRYPTED_GOODS_SENT)

ENCRYPTED_GOODS_SENT = ABORT || minc ?x -> (if x==epo then EPO_REC
                                              else ERROR)

EPO_REC = ABORT || (moutb !endorsedEpo -> ENDORSED_EPO_SENT)

ENDORSED_EPO_SENT = FAIL ||
    (minb ?x ->
     (if (x==paymentSlip) or (x==noPayment) then
         (FAIL ||
          moutc !x -> END)
        else ERROR))

```

Figure 3: The merchant process.

```

BANK = WAIT_ENDORSED_EPO

WAIT_ENDORSED_EPO = binm ?x ->
    (if x==endorsedEpo then
        (OK_TRANSACTION ||
        NOK_TRANSACTION)
    else WAIT_ENDORSED_EPO)

OK_TRANSACTION = debitC -> creditM ->
    boutm !paymentSlip -> FINAL_BANK(paymentSlip)

NOK_TRANSACTION = boutm !noPayment -> FINAL_BANK(noPayment)

FINAL_BANK(x) = binc ?y ->
    (if y==transactionEnquiry then
        boutc !x -> FINAL_BANK(x)
    else FINAL_BANK(x))

```

Figure 4: The bank process.

#### Check1 "SPEC1" "SYSTEM1"

`Check1` is a two argument command that determines whether its second argument is a failure/divergence refinement of its first.

Goods atomicity for NetBill is more complex and involves reasoning about the messages' send and receive synchronization events: `cinm.encryptedGoods` (this indicates *C*'s receipt of the message with the `encryptedGoods`), `cinm.paymentSlip` (this indicates *C*'s receipt of the `paymentSlip` message), and `cinb.paymentSlip` (this indicates *M*'s receipt of the `paymentSlip` message). Now, in order for *C* to "receive" the goods, *C* must receive both the `encryptedGoods` message and the encryption key (included in the `paymentSlip` message). We can now specify goods atomicity as follows:

```

SPEC2 =
  STOP ||
  (cinm.encryptedGoods ->
    (STOP ||
     (debitC ->
      creditM ->
        ((cinb.paymentSlip -> STOP) ||
         (cinm.paymentSlip -> STOP))))))

```

For simplicity, this specification is somewhat less general than our previous definition of goods atomicity (in particular, the above specification says that

goods must be paid for before they are received, whereas the previous definition stated that goods are received if and only if they are paid for and that the order of receipt and payment does not matter).

Our NetBill model satisfies both `SPEC1` and `SPEC2`; the full FDR code and excerpts from the model checking session appear in the Appendix.

### 3.2 A Simplified Digital Cash Protocol

The second example we investigate is a simplified digital cash protocol based on the offline Dicash protocols [3, 4]. (We have abstracted away a number of crucial components from the Dicash protocol.) In this protocol, electronic coins are withdrawn from and deposited into bank accounts and used for payments. Moreover, these payments are anonymous for the consumer, because coins are "blinded" during withdrawal. In contrast to NetBill, money atomicity for this protocol is non-trivial, because money is not centralized at the bank. We will focus our analysis on money atomicity. We do not provide a formal analysis of goods atomicity since the protocol actually violates this property (which we explain below).

#### The Protocol

Figure 5 contains our simplified digital cash protocol. The protocol consists of three parts: withdrawal

1.  $C \rightarrow B$  : *withdrawal request*
2.  $B \rightarrow C$  : *coin*
3.  $C \rightarrow M$  : *goods request with blinded coin*
4.  $M \rightarrow C$  : *challenge for the blinded coin*
5.  $C \rightarrow M$  : *response for the challenge*
6.  $M \rightarrow C$  : *goods*
7.  $M \rightarrow B$  : *blinded coin + response for the challenge*
8.  $B \rightarrow M$  : *deposit slip*

Figure 5: Simplified digital cash protocol.

of the coin (steps 1 – 2), spending of the coin (steps 3 – 6), and coin deposit (steps 7 – 8). We now describe each step in turn. The consumer  $C$  starts the protocol by requesting a withdrawal from the bank. The bank  $B$  responds with an electronic coin of the requested value. Before spending it,  $C$  “blinds” the coin to prevent the bank from tracing her payments. To spend the coin,  $C$  sends the coin to merchant  $M$ , and then responds to a challenge randomly selected by  $M$  (importantly,  $C$  maintains certain secret information about the coin so that only  $C$  can correctly respond to a random challenge).  $M$  locally verifies the consistency of the challenge/response pair, and then sends the goods. Finally,  $M$  deposits the coin by sending the coin and challenge/response pair to  $B$ , who responds with a deposit slip, assuming the coin is valid.

Observe that the essential part of spending the coin is not sending the coin to  $M$ , but responding to  $M$ ’s challenge. A consumer must take care not to respond to two different challenges for the same coin, because this will be considered evidence of fraudulent double spending: two challenge/response pairs for one coin are (with very high probability) sufficient for the bank to recover the identity of the consumer.

The protocol is clearly not goods atomic because  $M$  can omit step 6 but still deposit the coin. Also, note that the withdrawal part of the protocol (the first two messages) actually consists of a cut-and-choose protocol that involves a large number of message exchanges. These details are irrelevant for our analysis and are omitted.

### The Simplified Digital Cash Protocol in FDR

Figures 6, 7, and 8 present the consumer, merchant, and bank processes in FDR. To provide a more realistic modeling of the operation of the protocol, we have expanded the protocol behavior outlined in Figure 5 to include:

- coin returns: the consumer may choose to return coins to the bank for refund,
- fraud: the consumer and merchant can attempt double spending and multiple presentation of the same coin to the bank, and
- coin retention: the consumer may choose not to spend a coin and instead keep it for future use.

Unfortunately, in the context of this slightly more realistic system, a serious ambiguity arises. Consider the following scenario. A consumer withdraws a coin from the bank and attempts to use the coin to pay a merchant. However, as the consumer’s response to the merchant’s challenge was in transit, the communication network fails. The consumer is left in an uncertain situation. Has the coin been spent? If the merchant actually receives the response, then the consumer should consider the coin spent, but if not, then the coin is unspent. This is a critical issue for money atomicity, because if the consumer makes the wrong guess, then either money will be lost or she could be accused of double spending. To establish money atomicity, we allow the consumer to go to the bank in this situation and see if the coin has been spent; if it has not, she is eligible for a refund on the coin. Of course this leads to a problem: the consumer can spend the coin and then immediately go to the

bank and claim the coin may have been lost in transit and obtain a refund, and then moments later the merchant appears coin in hand. In practice this issue could be addressed by timeout/coin-lifetime management. In our model, we abstract the details of how this is solved and enter an “arbitration” state.

There are, however, two well-defined kinds of fraud that are detected and resolved in our model. The first is when a consumer attempts to double spend a coin, and the second is when a merchant attempts to deposit a coin twice. Both cases are detected by the bank and respective events `cFraud` and `mFraud` are triggered by the bank process. This is important, because it allows us to talk about money atomicity properties: in short, money atomicity holds when there are no `cFraud` and `mFraud` events. We elaborate further when we discuss money atomicity.

Our model includes only two challenge/response pairs, whereas there are really billions of possible such pairs. However, the specific identities of challenge/response pairs are immaterial: the critical property is the number of different challenges to which the customer responds (in fact there are only three important cases corresponding to zero, one, or more than one consumer response). Hence we consider just two “symbolic” challenge response pairs. We also abstract the statistical arguments, and simply state that if both of the symbolic challenge/response pairs are sent to the bank, then the bank has proof of consumer double spending.

We remark that the bank process is somewhat complicated because the bank must record information as it proceeds. This is somewhat cumbersome in FDR, and involves using process parameters. The main process involved here is `WAIT`, which has three parameters, the first indicating whether the coin has been deposited or returned, and the second and third indicating which challenge/response pairs have been seen.

### Money Atomicity for Simplified Digital Cash Protocol

Recall the money conservancy property given by the sum in Formula 1. The following CSP specification expresses this property in the context of the simplified digital cash protocol:

```
SPEC3 = STOP ||  
        (debitC -> ((depositC -> STOP) ||  
                      (cKeepsToken -> STOP) ||  
                      (depositM -> STOP)))
```

This specification holds in the presence of non-bank communication failures and limited non-bank agent failures. Surprisingly, it even holds in the presence of consumer and merchant fraud.

Next consider the cash property component of money atomicity. This states that possession of a coin gives the possessor the right to spend and/or deposit the coin. For  $C$ , this can be stated as:

```
SPECcashc =  
STOP ||  
(cinb.token -> ((tokenSpent -> STOP) ||  
                  (cKeepsToken -> STOP) ||  
                  (depositC -> STOP))).
```

and for  $M$  we have:

```
SPECcashm =  
STOP ||  
(mGetsToken ->  
((depositM -> STOP) ||  
 (mGetsRefundSlip -> STOP))).
```

$C$ 's cash property does in fact hold in the presence of fraud (that is, fraud by  $M$  cannot affect  $C$ 's cash property;  $M$  can fail to deliver the goods, but that is not a violation of the cash property, but of goods atomicity). However,  $M$ 's cash property does not hold: it can be violated by  $C$ 's fraud. When FDR is applied to this specification, it generates the following counterexample:

```
coutb.tokenReq, binc.tokenReq, debitC,  
boutc.token, cinb.token, coutm.goodsReq,  
minc.goodsReq, moutc.challengeA,  
cinm.challengeA, coutm.responseA,  
minc.responseA, mGetsToken, moutc.goods,  
moutb.responseA, binm.responseA, depositM,  
boutm.depositSlip, minb.depositSlip,  
cinm.goods, coutm.goodsReq, minc.goodsReq,  
moutc.challengeB, cinm.challengeB,  
coutm.responseB, minc.responseB, mGetsToken,  
moutc.goods, moutb.responseB,  
binm.responseB, boutm.alreadyDeposited,  
cFraud, minb.alreadyDeposited, cinm.goods,  
tokenSpent
```

This sequence of events corresponds to the scenario where a consumer double spends a coin: after finishing a successful transaction with the merchant (shown by events `mGetsToken`, `depositM`, and `cinm.goods`), the consumer uses the coin again (`mGetsToken`), gets the goods (`cinm.goods`), but instead of successfully

```

CONSUMER = ABORT || (coutb !tokenReq -> TOKEN_REQ_SENT)

TOKEN_REQ_SENT = cinb ?x ->
                  if x==token then (USE_TOKEN [] RETURN_TOKEN [] KEEP_TOKEN)
                  else ERROR

USE_TOKEN = coutm !goodsReq -> GOODS_REQ_SENT

KEEP_TOKEN = cKeepsToken -> END

GOODS_REQ_SENT = cinm ?x -> (if (x==challengeA) then
                                  (coutm !responseA -> TOKEN_USED)
                                  else (if (x==challengeB) then
                                         (coutm !responseB -> TOKEN_USED)
                                         else RETURN_TOKEN)) []
                                  timeoutEvent -> RETURN_TOKEN

TOKEN_USED = (cinm ?x ->
              (if x==goods then (tokenSpent -> C_MAY_BE_FRAUD)
               else RETURN_TOKEN)) []
              (timeoutEvent -> RETURN_TOKEN)

RETURN_TOKEN = coutb !token -> cinb ?x ->
                 (if x==refundSlip then REFUND RECEIVED
                  else if x==depositSlip then (tokenSpent -> ARBITRATION)
                  else ERROR)

C_MAY_BE_FRAUD = END || USE_TOKEN

```

Figure 6: The consumer process for the simplified digital cash protocol.

```

MERCHANT = ABORT || WAITING_GOODS_REQ(None)

WAITING_GOODS_REQ(previousResponse) = minc ?x ->
  (if (x==goodsReq) then
    (if previousResponse==None then (CHALLENGE_A [] CHALLENGE_B)
     else if previousResponse==responseA then CHALLENGE_B
     else CHALLENGE_A)
   else ERROR)

CHALLENGE_A = moutc !challengeA-> WAIT_FOR_RESPONSE(responseA)

CHALLENGE_B = moutc !challengeB-> WAIT_FOR_RESPONSE(responseB)

WAIT_FOR_RESPONSE(response) =
  minc ?x -> (if x==response then (mGetsToken -> SEND_GOODS(x))
                else moutc !badResponse -> NO_TRANSACTION)

SEND_GOODS(response) = (moutc !goods -> DEPOSIT_TOKEN(response)) []
                      DEPOSIT_TOKEN(response)

DEPOSIT_TOKEN(response) = moutb !response -> WAIT_FOR_BANK(response)

WAIT_FOR_BANK(response) = minb ?x ->
  (if x==depositSlip then M_MAY_BE_FRAUD(response)
   else if x==refundSlip then (mGetsRefundSlip -> STOP)
   else if x==alreadyDeposited then FRAUD_DISCOVERED
   else ERROR)

M_MAY_BE_FRAUD(response) = END ||
                           DEPOSIT_TOKEN(response) ||
                           WAITING_GOODS_REQ(response)

```

Figure 7: The merchant process for the simplified digital cash protocol.

```

BANK = binc ?x ->
    (if x==tokenReq then (boutc !badBalance -> STOP []
                           debitC -> boutc !token -> WAIT(0, 0, 0))
     else ERROR)

WAIT(cashedFlag, responseA, responseB) =
  binc ?x ->
    (if (x==token) then
      (if (cashedFlag==0) then
          (depositC -> boutc !refundSlip -> WAIT(1, 0, 0))
        else if (responseA==1 or responseB==1) then
          (arbitration -> boutc !depositSlip ->
           WAIT(cashedFlag, responseA, responseB))
        else WAIT(cashedFlag, responseA, responseB)))
     else WAIT(cashedFlag, responseA, responseB)) []
  binm ?x ->
    (if (x==responseA) then
      (if (cashedFlag==0) then
          (depositM -> boutm !depositSlip ->
           WAIT(1, 1, responseB))
        else if (responseA==1) then
          (boutm !alreadyDeposited -> mFraud ->
           WAIT(cashedFlag, responseA, responseB))
        else if (responseB==1) then
          boutm !alreadyDeposited-> cFraud ->
          WAIT(cashedFlag, responseA, responseB)
        else (arbitration -> boutm !refundSlip ->
              WAIT(cashedFlag, responseA, responseB)))
      else if (x==responseB) then
        (if (cashedFlag==0) then
          (depositM -> boutm !depositSlip ->
           WAIT(1, responseA, 1))
        else if (responseB==1) then
          (boutm !alreadyDeposited -> mFraud ->
           WAIT(cashedFlag, responseA, responseB))
        else if (responseA==1) then
          (boutm !alreadyDeposited -> cFraud ->
           WAIT(cashedFlag, responseA, responseB))
        else (arbitration -> boutm !refundSlip ->
              WAIT(cashedFlag, responseA, responseB)))
      else WAIT(cashedFlag, responseA, responseB))
    )

```

Figure 8: The bank process for the simplified digital cash protocol.

depositing the coin, the merchant receives a message indicating that the coin in question had been spent before (`boutm.alreadyDeposited`), and consumer fraud (`cFraud`) is revealed. (For further details, see the Appendix.)

However, in the absence of revealed fraud (i.e. in the case where there are no (`cFraud`) or (`mFraud`) events),  $M$ 's cash property is satisfied. We express this as follows:

```
SPECcashm' =
STOP ||
((mGetsToken ->
  ((depositM -> STOP) ||
   (mGetsRefundSlip -> STOP))) ||
FRAUD)
```

where `FRAUD` denotes processes that contain at least one fraud event, `cFraud` or `mFraud`, and are otherwise arbitrary. Using FDR we checked that indeed the unmodified protocol satisfies this modified property.

## 4 Summary and Discussion of Our Contributions

We have presented a model checking approach for verifying atomicity properties of electronic commerce protocols. Until now, such properties have been reasoned about using informal and *ad hoc* methods. However, these methods have not been adequate and numerous significant errors have been made in the design of electronic commerce protocols. Not only are the protocols themselves moderately complex and subtle, but the properties expected and/or desired are often only partially specified and not fully understood. Model checkers can address both aspects of this problem: we can write precise definitions of the behavior of a protocol (at any desired level of abstraction) and then formulate protocol properties and test that they are satisfied. If a property is not satisfied, a model checker will give a counterexample, which we can use to step through the execution of the protocol to better understand its behavior. This kind of interactive experimentation is a very powerful tool for debugging and modifying both protocols and the properties we expect to hold. In our experience, the most obvious specification of a property is often incorrect or inadequately expresses the property, and that by experimenting, we frequently obtain more precise and stronger properties.

We have discussed here two properties: money atomicity and goods atomicity. We believe that these techniques will extend to other properties such as anonymity, transactional properties (consistency, isolation, durability), nonrepudiation, certified delivery, etc. Similarly, though we demonstrated our approach on only two protocols, they are radically different from each other; model checking atomicity and other properties should easily be applicable to other electronic commerce protocols.

In our modeling of NetBill and the simple digital cash protocol, we have employed a number of abstractions. For example, we have ignored the low-level details of the the underlying cryptographic mechanisms and just treated them as a blackbox (this is the standard "perfect encryption" assumption). In fact our model goes one step further: we have chosen not to even mention encryption/decryption/signature operations so that we could develop as simple a model as possible and focus on atomicity properties.

A second example of a reasonable abstraction we applied is in modeling the challenge/response pairs in the electronic cash protocol: many billions of different pairs were represented as just two pairs. As a third example, recall that we modeled NetBill and the electronic cash protocol assuming just a small number of players: there was only one bank, one consumer, and one merchant; moreover, we consider only one run of the protocol. In practice, we would expect these protocols to be used in huge networks with large numbers of consumers, merchants and banks, with multiple interleaved runs of the protocols. Roscoe and McCarthy justify similar simplifications in their work using FDR to model check *data-independent* properties of concurrent processes [20].

In summary, finding the right abstractions is essential to finding an effective representation of a protocol for model checking. The goal is to map an intractable problem into a tractable abstracted problem in such a way that proving something about the abstracted problem says something meaningful about the real problem at hand.

## 5 Future Work

We plan to investigate some of the assumptions made and abstractions used in our modeling of NetBill and the electronic cash protocol. For example, suppose we consider multiple merchants, consumers, and banks? Multiple runs of a protocol? Multiple transaction

values? What number (of players, runs, values) is too large for current model checking technology to handle?

Could we provide a formal justification for some of the abstractions we used? For example, can we prove that if goods atomicity holds for one merchant and one consumer, then it holds for multiple merchants and consumers? We treated the cryptographic component of the protocol as orthogonal to our analysis for atomicity. We doubt that analyzing an enriched FDR model to include the cryptographic component would be tractable; however, we believe that it may be possible to use other model checking methods to address some cryptographic aspects. Then, we may be able to factor the problem of whether, say, NetBill is goods atomic, into two problems: (a) determining whether the cryptographic aspects of NetBill are secure, and (b) determining whether Netbill is goods atomic, assuming its cryptographic aspects are secure (which is essentially what we have proved in this paper).

Finally, we plan to provide more comprehensive failure modeling. In this paper we have used the following informal principle: failures by one agent can interfere with that agent's atomicity properties, but they must not interfere with another agent's properties. We can formulate this in a more precise manner as follows. First, we analyze the atomicity properties that we wish to establish, and associate components of these properties with individual agents. For example, goods atomicity can be stated as: "if the consumer pays then the consumer gets the goods" and "if the consumer gets the goods then the merchant gets paid". The first part of this statement is the consumer's property, and the second is the merchant's. Then, for each agent, we consider a model in which the agent does not fail but other agents fail arbitrarily, and we seek to establish those components of the atomicity properties associated with the non-failed agent. Even more ambitiously, limited bank failure is another important—and realistic—aspect to model for future work.

## References

- [1] Ross Anderson. UEPS - a second generation electronic wallet. In *The Second European Symposium on Research in Computer Security*, pages 411–418, 1992.
- [2] L. Jean Camp, Marvin Sirbu, and J. D. Tygar. Token and notational money in electronic commerce. In *Proceedings of the First USENIX Workshop in Electronic Commerce*, pages 1–12, July 1995.
- [3] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.
- [4] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology — CRYPTO '88 Proceedings*, pages 200–212. Springer-Verlag, 1990.
- [5] Benjamin Cox, J. D. Tygar, and Marvin Sirbu. NetBill security and transaction protocol. In *Proceedings of the First USENIX Workshop in Electronic Commerce*, pages 77–88, July 1995.
- [6] David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525, 1992.
- [7] J. Gray and A. Reuter. *Transaction Processing*. Morgan-Kauffman, 1993.
- [8] James N. Gray. A transaction model. Technical Report RJ2895, IBM Research Laboratory, San Jose, California, August 1980.
- [9] Zvi Har'El and Robert P. Kurshan. Software for analytical development of communications protocols. In *AT&T Technical Journal*, pages 45–60, January/February 1990.
- [10] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, New Jersey, 1985.
- [11] G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
- [12] Daniel Jackson. Nitpick: A checkable specification language. In *Proc. Workshop on Formal Methods in Software Practice*, San Diego, CA, January 1996.
- [13] Rajashekhar Kailar. Reasoning about accountability in protocols for electronic commerce. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 236–250, May 1995.

- [14] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems: Second International Workshop, TACAS '96*, pages 147–166, March 1996.
- [15] Formal Systems (Europe) Ltd. *Failures Divergence Refinement—User Manual and Tutorial*, 1993. Version 1.3.
- [16] Nancy Lynch, Michael Merritt, William Weihl, and Alan Fekete. *Atomic Transactions*. Morgan Kaufmann, San Mateo, CA, 1994.
- [17] K. L. McMillan. Symbolic model checking: An approach to the state explosion problem. Technical Report CMU-CS-92-131, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1992. Ph.D. thesis.
- [18] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978. Also Xerox Research Report, CSL-78-4, Xerox Research Center, Palo Alto, CA.
- [19] A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings 1995 IEEE Symposium on Security and Privacy*, pages 114–127, May 1995.
- [20] A.W. Roscoe and H. MacCarthy. A case study in model-checking CSP. submitted for publication, October 1994.
- [21] Marvin Sirbu and J. D. Tygar. NetBill: an internet commerce system optimized for network delivered services. *IEEE Personal Communications*, pages 34–39, August 1995.
- [22] J. D. Tygar. Atomicity in electronic commerce. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 8–26, May 1996.
- [23] Jeannette Wing and Mandana Vaziri-Farhani. A case study in model checking software systems. *Science of Computer Programming*, January 1996. To appear. Preliminary version in *SIGSOFT Proc. of Foundations of Software Engineering*, October 1995.

## A Appendix

We present the full code for our model of NetBill and the digital cash protocol (lines beginning with “–” are comments). We also give some excerpts of the FDR verifications.

### A.1 NetBill

```
-- The data types DATAxy are the set of data
-- that are transmitted from the principal x
-- to the principal y;
DATAmc = {noPayment, paymentSlip,
           encryptedGoods}
DATAbc = {noPayment, paymentSlip, noRecord}
DATAcm = {goodsReq, epo}
DATAcb = {transactionEnquiry}
DATAbm = {paymentSlip, noPayment}
DATAmb = {endorsedEpo}

-- The names of the channels are of the form
-- x(in/out)y, where in/out refers to the
-- direction (relative to x), and y is the
-- other party of the communication;
pragma channel coutm : DATAcm
pragma channel coutb : DATAcb
pragma channel moutc : DATAmc
pragma channel moutb : DATAmb
pragma channel boutc : DATAbc
pragma channel boutm : DATAbm
pragma channel minc : DATAcm
pragma channel binc : DATAcb
pragma channel cinm : DATAmc
pragma channel binm : DATAmb
pragma channel cimb : DATAbc
pragma channel minb : DATAbm

pragma channel creditM, debitC, timeoutEvent

[The consumer process CONSUMER:
as given in the paper.]

[The merchant process MERCHANT:
as given in the paper.]

[The bank process BANK:
as given in the paper.]

SUCCESS = STOP
ERROR = STOP
```

```

NO_FUNDS = STOP
NO_TRANSACTION = STOP
END = STOP
ABORT = STOP
FAIL = STOP

-- The communication channels: only those
-- involving the bank are reliable.

COMMcm = []x: DATAcm @
           (coutm ?x ->
            (COMMcm [] 
             (minc !x -> COMMcm)))
COMMcb = []x: DATAbc @
           (coutb ?x ->
            ((binc !x -> COMMcb)))
COMMmc = []x: DATAmc @
           (moutc ?x ->
            (COMMmc [] 
             (cinm !x -> COMMmc)))
COMMmb = []x: DATAmb @
           (moutb ?x ->
            ((binm !x -> COMMmb)))
COMMbc = []x: DATAbc @
           (boutc ?x ->
            ((cinb !x -> COMMbc)))
COMMbm = []x: DATAbm @
           (boutm ?x ->
            ((minb !x -> COMMbm)))

COMM = COMMcm [|{}|] COMMcb [|{}|] COMMmc
          [|{}|] COMMmb [|{}|] COMMbc
          [|{}|] COMMbm

-- The Whole Netbill System

CIO = {| coutm, coutb, cinm, cinb |}
MIO = {| moutc, moutb, minc, minb |}
BIO = {| boutc, boutm, binc, binm |}
BINT = {debitC, creditM}
BTOT = union(BIO, BINT)
COMMIO = union(CIO, union(MIO, BIO))
COMMIO' = diff(COMMIO,
               {cinm.encryptedGoods,
                cinm.paymentSlip,
                cinb.paymentSlip})

SYSTEM1 =
((CONSUMER [|{}|] MERCHANT [|{}|] BANK)
 [| COMMIO |] COMM)
\ union(COMMIO, {timeoutEvent})

```

```

SYSTEM2 =
((CONSUMER [|{}|] MERCHANT [|{}|] BANK)
 [| COMMIO |] COMM)
\ union(MIO,
        union(BIO,
              union({| coutm, coutb |},
                     {cinm.noPayment,
                      cinb.noPayment,
                      cinb.noRecord,
                      timeoutEvent})))

```

```

SPEC1 = STOP | ~|
((debitC -> creditM -> STOP) []
 (creditM -> debitC -> STOP))

```

```

SPEC2 = (STOP | ~|
(cinm.encryptedGoods -> (STOP | ~|
(debitC -> creditM ->
((cinb.paymentSlip -> STOP) | ~|
(cinm.paymentSlip -> STOP))))))

```

The check of SPEC1 generated the following FDR output:

```

fdr> Check1 "SPEC1" "SYSTEM1";
SPEC1 ....(5 states)
CONSUMER .....(10 states)
MERCHANT .....(15 states)
BANK .....(10 states)
COMMcm .(3 states)
COMMcb .(2 states)
COMMmc .(4 states)
COMMmb .(2 states)
COMMbc .(4 states)
COMMbm .(3 states)
readalphabet:
Alphabet contains 3 events [up to 998]
6 configuration masks in 6 transitions
3
5 reachable configurations
0
nfcompact :
compacting 4 state normal form:
now 4 states
87 configuration masks in 67 transitions
139
199 reachable configurations
1 The implementation does indeed
refine the normal form.
Checked 199 pairs.
Refinement check succeeded

```

```
No failure in this context!
val it = - : (label,selector,cause) Context
fdrt>
```

## A.2 A Simplified Digital Cash Protocol

```
-- The data types DATAxy are the set of data
-- that are transmitted from the principal x
-- to the principal y;
DATAcb = {tokenReq, token}
DATAcm = {responseA, responseB, goodsReq}
DATAbc = {token, badBalance, badToken,
          depositSlip, refundSlip}
DATAbm = {refundSlip, depositSlip,
          alreadyDeposited}
DATAmc = {goods, badResponse, challengeA,
          challengeB}
DATAmb = {responseA, responseB}

[Communication channels:
as given for NetBill.]

pragma channel goodsReceived, debitC,
              depositC, depositM, mFraud, cFraud,
              mGetsRefundSlip, tokenSpent,
              mGetsToken, cKeepsToken,
              timeoutEvent, arbitration

[The consumer process CONSUMER:
as given in the paper.]

[The merchant process MERCHANT:
as given in the paper.]

[The bank process BANK:
as given in the paper.]

ABORT = STOP
END = STOP
REFUND_RECEIVED = STOP
ERROR = STOP
ARBITRATION = arbitration -> STOP
NO_TRANSACTION = STOP
FRAUD_DISCOVERED = STOP

[COMM: as given for NetBill.]

-- The communication events
```

```
CIO = [| coutm, cinm, coutb, cinb |]
MIO = [| moutc, minc, moutb, minb |]
BIO = [| boutc, boutm, binc, binm |]
COMMIO = union(CIO, union(MIO, BIO))
COMMIO' = diff(COMMIO, {cinb.token})

-- The model for Money Atomicity

SYSTEM3 =
((CONSUMER [|{}|] MERCHANT [|{}|] BANK)
 [| COMMIO |] COMM)
\ union(COMMIO,
      {goodsReceived, mGetsRefundSlip,
       mGetsToken, tokenSpent, mFraud,
       cFraud, timeoutEvent,
       arbitration})

SYSTEMc =
((CONSUMER [|{}|] MERCHANT [|{}|] BANK)
 [| COMMIO |] COMM)
\ union(COMMIO',
      {goodsReceived, debitC,
       depositM, mGetsToken,
       mGetsRefundSlip, timeoutEvent,
       arbitration, mFraud, cFraud})

SYSTEMm =
((CONSUMER [|{}|] MERCHANT [|{}|] BANK)
 [| COMMIO |] COMM)
\ union(COMMIO,
      {goodsReceived, debitC, depositC,
       tokenSpent, cKeepsToken,
       timeoutEvent, arbitration,
       mFraud, cFraud})

SYSTEMm' =
((CONSUMER [|{}|] MERCHANT [|{}|] BANK)
 [| COMMIO |] COMM)
\ union(COMMIO,
      {goodsReceived, debitC, depositC,
       tokenSpent, cKeepsToken,
       timeoutEvent, arbitration})

SPEC3 =
STOP |||
(debitC -> ((depositC -> STOP) |||
(cKeepsToken -> STOP) |||
(depositM -> STOP)))

SPECcashc =
STOP |||
(cinb.token -> ((tokenSpent -> STOP) |||
```

```

(cKeepsToken -> STOP) |~|
(depositC -> STOP)))
SPECcashm =
STOP |~|
(mGetsToken ->
((depositM-> SPECcashm) |~|
(mGetsRefundSlip -> SPECcashm)))

SPECcashm' =
STOP |~|
((mGetsToken ->
((depositM-> STOP) |~|
(mGetsRefundSlip -> STOP)))
|~| FRAUDM)

FRAUDM = (mGetsToken -> FRAUDM) |~|
(mGetsRefundSlip -> FRAUDM) |~|
(depositM -> FRAUDM) |~|
(mFraud -> ANYM) |~|
(cFraud -> ANYM)

ANYM = (mGetsToken -> ANYM) |~|
(mGetsRefundSlip -> ANYM) |~|
(depositM -> ANYM) |~|
(mFraud -> ANYM) |~|
(cFraud -> ANYM)
STOP

```

The check of SPEC1 generated the following FDR output (and counter-example):

```

fdr> Check1 "SPECcashm" "SYSTEMm";
SPECcashm ....(6 states)
CONSUMER .....(16 states)
MERCHANT .....(24 states)
BANK .....(19 states)
COMMcm .(4 states)
COMMcb .(3 states)
COMMmc .(5 states)
COMMmb .(3 states)
COMMbc .(6 states)
COMMbm .(4 states)
readalphabet:
Alphabet contains 4 events [up to 998]
7 configuration masks in 7 transitions
3
6 reachable configurations
0
nfcompact :
compacting 2 state normal form:

```

```

now 2 states
174 configuration masks in 149 transitions
120
1002 reachable configurations
1

SYSTEMm
Interface={|{|{depositM,mGetsRefundSlip,
mGetsToken,tick|}|}}
has behaviour
After <tau,tau,tau,tau,tau,tau,tau,tau,
tau,tau,tau,tau,mGetsToken,tau,tau,
tau,depositM,tau,tau,tau,tau,
tau,tau,tau,tau,tau,tau,mGetsToken,
tau,tau,tau,tau,tau,tau,tau,tau>
refuses {|{|{depositM,mGetsRefundSlip,
mGetsToken,tick|}|}}
Accepts only {{|||}}
Contributions:
Component 1
((CONSUMER[|||]MERCHANT)
[|||]BANK)
[||{|coutb,coutm,moutc,moutb,boutc, boutm,
cinb,cinm,minc,minb,binc,binm|}|]COMM
Interface={|{|{coutb,coutm,moutc,moutb,boutc,
boutm,cinb,cinm,minc,minb,
binc,binm,debitC,depositC,
depositM,mFraud,cFraud,
mGetsRefundSlip,tokenSpent,
mGetsToken,cKeepsToken,
timeoutEvent,arbitration,
tick|}|}
has behaviour
After <tau,tau,coutb.tokenReq,binc.tokenReq,
debitC,boutc.token,cinb.token,
coutm.goodsReq,minc.goodsReq,
moutc.challengeA,cinm.challengeA,
coutm.responseA,minc.responseA,
mGetsToken,moutc.goods,
moutb.responseA,binm.responseA,
depositM,boutm.depositSlip,
minb.depositSlip,tau,cinm.goods,tau,
coutm.goodsReq,minc.goodsReq,
moutc.challengeB,cinm.challengeB,
coutm.responseB,minc.responseB,
mGetsToken,moutc.goods,
moutb.responseB,binm.responseB,
boutm.alreadyDeposited,cFraud,
minb.alreadyDeposited,cinm.goods,tau,
tokenSpent>
refuses {|{|{coutb,coutm,moutc,moutb,boutc,

```

```
boutm,cinb,cinm,minc,minb,binc,
binm,debitC,depositC,depositM,
mFraud,cFraud,mGetsRefundSlip,
tokenSpent,mGetsToken,
cKeepsToken,timeoutEvent,
arbitration,tick|{}|}

Accepts only {{|||}|
it = - : (label,selector,cause) Context
fdr>
```

# BigDog: Hierarchical Authentication, Session Control, and Authorization for the Web

Benjamin Fried & Andrew Lowry

*Morgan Stanley & Co, Inc.*

## Abstract

As part of Morgan Stanley's clients-only web site, we have built a demon that provides sophisticated authentication, authorization, and session-control services to web servers. Authentication techniques are ordered "hierarchically" by our belief in the technique used, and authentication requirements for access can be specified on a per-user, per-page basis.

## 1 Introduction

Morgan Stanley is a global financial services firm with a large and diversified clientele. The firm offers a wide variety of services—debt, foreign exchange, equities and derivatives trading, institutional and personal asset management, custodial services, and research, to name a few. Our clients are businesses, governments, and individuals of very high net worth. After building a large internal web in 1994 and an external web site in 1995, in January of 1996 we began planning a private, clients-only web site, where we would offer highly customized, client-specific web applications. The variety of applications and clients we planned for this web site led us to develop a hierarchical authentication and authorization model for our web services, which we implemented in a demon called BigDog.

## 2 Security Model

The firm provides a huge variety of services, some of which are much more sensitive than others—for example, we might place a greater value on the privacy of a client's portfolio statements than on a six-month-old report on Myanmar's timber industry. Security measures that are proper and reasonable for a very sensitive transaction might be needlessly cumbersome for another, far less sensitive transaction. To further com-

plicate things, security requirements for the same application often vary from client to client.

The types of threats we worried about varied from client to client and from application to application. In some cases, clients would have private leased lines connected directly to our firewalls. In other cases, they would be coming to our web site over the Internet, from their own Internet site or via a service provider. Some clients' computing facilities meet rigorous security standards; others are far less concerned about internal security. Some clients have many users, each allowed to use different sets of applications on our web site, and all sharing a single workstation. Our primary concerns (in no particular order) were

- Users at the same site using each others' applications (e.g., a CFO might be entitled to see daily profit and loss statements, but an administrative assistant might only be allowed to see research).
- Users at different sites accessing each others' applications. Many of our clients are direct competitors, and something like this would be disastrous.
- Bandits on the Internet stealing sensitive data. The data we provide varies greatly in sensitivity, but we do not want even the least sensitive of it to be *easily* stolen. Even if such a theft did not directly harm us or one of our clients, the negative publicity it might create could severely damage the firm's image.

An out-of-the-box approach to web security might be to use the web server's built-in authentication/authorization scheme (typically based on password files readable by the web server) and to use an encrypted transport.

However, given our security concerns, the generic scheme was inadequate. Instead, we came up with a web security model that encompasses encryption, user name conventions, authentication, authorization, and controls on web sessions.

## 2.1 Encryption

All client traffic to the web site uses the HTTP-over-SSL (“https”) protocol. This includes all authentication messages between web server and client: user IDs and passwords *never* go over the net in the clear.

SSL provides us with several desirable features:

- Traffic text not be easily observed.
- Clients can be assured that they are talking to Morgan Stanley’s secure web site, and not an impostor that has somehow taken our name or address.
- The Integrity of the traffic between our site and the clients’ site can be guaranteed.

Although encryption is a critical element of our security model, it is outside of the realm of the BigDog authentication and authorization server, which actually does not know about the encryption of the channel between web server and client.

## 2.2 Identity

For our purposes, the common conventions for user names on most limited-access web sites were inadequate. In those cases, a user ID and password are unique to the web site—for example, one might have completely different user names at the *Wall Street Journal* on-line, at *Hotwired*, and at the Netscape developers’ site. We wanted user names to convey our recognition of the client’s own identity, and not force them to adopt a new identity just in dealing with Morgan Stanley. BigDog user names are an email address, in the ubiquitous `username@domainname` form. They tell us how the client identifies him/herself — `barney_rubble@foo.com`, `wilma_flintstone@bar.com`, and so on.<sup>1</sup> They also avoid conflicts in user IDs chosen by clients—we don’t have to be in the position of telling an important client that his or her preferred user ID is already taken. The additional `@foo.com` is rarely a burden; if the user did not type in a domain name when entering his or her user ID, BigDog will automatically append the domain name of the browser host.

When clients do not have their own Internet domain, we create usernames for them in the same form, but with a made-up domain name, and encourage them to register that domain with the InterNIC.

<sup>1</sup>When one goes to Netscape’s download site, the registration process creates a user ID of the form `username@domainname`, in just this way.

## 2.3 Multiple means of authentication

Security needs vary greatly depending on the sensitivity of the application and the particulars of the client. In some cases we must be more sure of the client’s identity than others. This concept is central to our design, and led to an authentication system with many types of authentication, of varying degrees of certainty.

In BigDog, authentication is not a binary decision. Instead, each authentication technique is associated with a certainty level; the higher the level, the more certain the identity of the user. Access decisions are based on the certainty level of the authentication technique used. The certainty level is also passed on to cgi-bin programs, so that they can use it as the basis for further decisions on access control or presentation method for information returned.

So far, we have implemented four types of authentication:

- Username of client without any sort of verification of identity
- Username and BigDog-maintained reusable password
- Username and Kerberos password<sup>2</sup>
- Username, one-time password and PIN (we use Security Dynamics’ SecurID system)

In all cases, if the IP address of the browser matches our record of the client’s “home domain,” the authentication certainty level is incremented by a constant to indicate the added certainty that knowledge gives us. Note that the home domain does not have to be the same as the domainname contained in the client’s user ID, and users do not have to have a home domain at all (this might be the case for a client that uses an ISP or on-line service for their Internet connectivity). Table 1 illustrates how these authentication techniques map to certainty levels. The authentication certainty values we use are completely arbitrary, although the values can appear in access control lists, they are only important in how they relate the authentication techniques to one another.

This model is open-ended; adding a new authentication technique just requires deciding on the certainty level of the technique, adding an entry in a table, and linking in the necessary authentication routine. In particular, we are planning for the emergence of client public key certificates. At the time that BigDog was designed, web browsers did not implement client side certificates. Furthermore, there were a host of issues like

<sup>2</sup>Not currently used; we do not do inter-realm authentication.

<i>Authentication Type</i>	<i>certainty level</i>	<i>certainty when at home domain</i>
None	0	n/a
User ID	10	15
User ID + Password	20	25
User ID + Password + One Time Password	30	35
<i>User ID + Client Certificate</i> <sup>a</sup>	40	45

<sup>a</sup>Not yet implemented.

Table 1: Sample Authentication Certainty Levels

key sizes, certificate authority trust policies, certificate protection standards, and key escrow that had to be resolved both within Morgan Stanley and in the standards-setting bodies before this technology could be incorporated into BigDog.

## 2.4 Authentication-sensitive access control

BigDog uses its own database of access-control lists (ACLs), governing authorization for the web sites it protects. An ACL contains

- a possibly wildcarded URL
- a possibly wildcarded username
- the minimum authentication certainty level that is required in order to present the page to that user
- an optional set of attribute-value pairs to be inserted into the environment of cgi scripts protected by this ACL

Authentication is hierarchical: someone authenticating at level  $x$  is entitled to see all pages available to them when authenticated at level  $x$  or any level below. If the user has authenticated at a level lower than  $x$ , and wishes to see a page protected at level  $x$ , BigDog tells the web server to reauthenticate the user at the necessary level. An example of the kind of control this gives us is that, we can implement access rules like “client c@foo.com only has to present his user ID and multi-use password to read research reports when connecting from the office, but if he is coming to our web site from an on-line service or an account at an ISP he’ll have to authenticate with SecurID.”<sup>3</sup>

Per-ACL attribute-value pairs are not used in BigDog’s authorization decisions, but are used to allow cgi programs to make additional decisions at an even finer level of granularity.

<sup>3</sup>That is, c@foo.com must authenticate at a certainty level of 25 or more.

## 2.5 Sessions

There’s a fundamental problem in trying to fit one-time passwords into the HTTP protocol. Protected web pages that have to be served in multiple HTTP requests (e.g. a page which includes images) require a separate authentication for every component. It’s ridiculous to make a user authenticate with a different one-time password for each piece of a document. In order to make one-time passwords usable, we had to be able to specify a lifetime for the password, during which it could be used repeatedly.<sup>4</sup> We also wanted to be able to force reauthentication after some amount of inactivity, and we wanted to be able to set hard limits on the maximum length of time between authentications.

In BigDog, all client interactions are part of a session. Each session has associated with it

- idle timeout: the maximum number of seconds of inactivity between user requests; if exceeded, the user must reauthenticate
- session timeout: the maximum amount of time that can pass before the user reauthenticates
- user’s current authentication level
- last password the user authenticated with
- client host name
- client IP address

Limits on idle time and session length are configured on a per-user basis, and when one of these parameters is exceeded, BigDog instructs the web server to make the user reauthenticate before any more requests will be serviced.

When a client first authenticates to a BigDog-secured web server, BigDog assigns a unique *session ID* to the user. We use client cookies[1] to store an opaque version of the session ID in the user’s web browser. If the browser supports cookies, the opaque session ID will

<sup>4</sup>To be precise, ours are actually *use-a-few-times* passwords

be sent to our web server with every request from that user's browser. Note that Session identifiers are not directly used for authentication: all web page access are still authenticated, but the cookie serves ■ a hint to help BigDog figure out which session this interaction is a part of.

When a web browser does not support cookies, BigDog attempts to deduce which session the current request is part of, using the user name, host name and IP address, and password. This heuristic is conservative, and will not assign ■ request to the wrong session, though it may fail to deduce the session of a request that is part of an ongoing session.

At the outset, we did not consider cookies to be trustworthy enough to use in place of authentication. We were worried that cookies might end up in text files on file servers, or that they could be shoulder surfed by browsers that ask the user if he/she wants to accept the cookie. It seemed best to make due with the extra HTTP round trip for authentication, and wait for the web community to develop more practical experience in the use of cookies as authenticators.

Even though we don't use session identifiers to short circuit the authentication process, they still have their uses. Sessions give us an easy way to impose idle timeouts and force periodic reauthentication, and are useful when trying to analyze web logs, especially when all the users at one client site come to our web site through a proxy server.

## 2.6 User Name completion

No one likes typing more than they have to. If a client comes to us from a browser within their home domain, they do not have to enter a fully-qualified user name. Whenever BigDog gets a username without a domain name component, it automatically appends the domain name of the client browser before authenticating the user.

## 3 How BigDog Affects the Flow of Control of ■ Web Server

In a typical web server not secured by BigDog, the logical flow of servicing a request for a protected document looks something like this:

1. Request comes in from client:  
GET /blah http/1.0.
2. Web server determines that URL /blah is protected.

3. Did request include Authorization: header? If so, validate the user ID and password; if successful, serve the document. Otherwise proceed.
4. Server requests authentication, requesting identification within a particular "authentication domain."
5. Browser prompts user for user ID and password, using the "authentication domain" the server returned.
6. Browser sends back same GET request, but with the Authorization: header containing the base64-encoded user ID and password.
7. Return to step 1

In a BigDog-protected web server, the flow of control is similar, however, steps two and three are combined, and instead of the web server determining that the URL is protected, the web server passes the request, any authorization information, and the host name of the client browser to BigDog. BigDog performs authentication and authorization checks, and can instruct the web server to

- Set a browser cookie containing session identification information
- Set a number of environment variables in processes invoked by the web server, including the user ID, personal name, and current authentication level of the user
- Grant access to the URL
- Ask the user to reauthenticate at a higher certainty level
- Deny the existence of the document

Figure 1 shows a typical interaction between a client web browser, ■ web server, and BigDog. The user initially authenticates at a level too low to get the document in question, and must reauthenticate at a higher level in order to get the document.

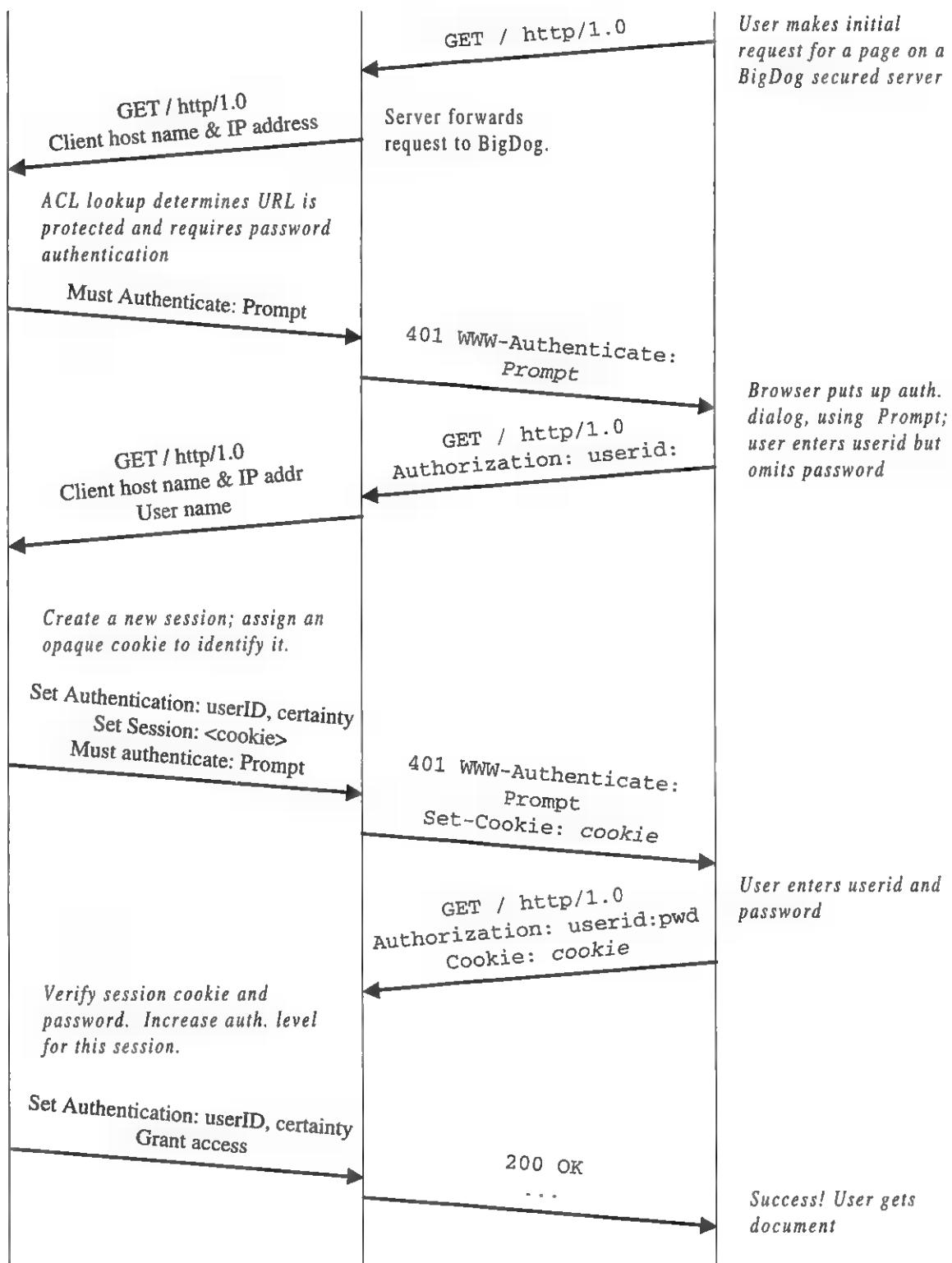
## 4 BigDog Implementation

### 4.1 The NSAPI: The Good, the Bad, and the Ugly

We originally planned to implement our security model as extensions to Netscape's secure web server, through the Netscape Server API ("NSAPI") extension mechanism. It breaks the process of processing an HTTP request into six steps.[2]



Client Web browser



1. Authentication
2. Name translation
3. Path Checks (including authorization)
4. Object type (loading an object and binding a type to it)
5. Service (transmitting the object to the browser)
6. Logging

The NSAPI allows the user to insert new modules at any one of those six points, or to add a new routine to be called at server initialization time.

After quite a bit of initial enthusiasm, we became frustrated with using the NSAPI to embed a large subsystem into the web server. The NSAPI was not very well documented at the time we started writing code for it,<sup>5</sup> and there were no substantial, useful examples for us to base our code on.<sup>6</sup> Furthermore, in Netscape's early web servers, the NSAPI had some frustrating behaviors. For example, user IDs would lose their @domain.name component between the path check and logging steps.

We gave up on writing our authentication and authorization system exclusively in the NSAPI when vendor-supplied libraries for one of our authentication systems could not run in the same process space as the Netscape server. This was not the fault of NSAPI: the vendor in question produces a great authentication product, but the library makes liberal use of global variables, and it did not fit well into the multi-threaded model of the Netscape server.

In any event, it was clear that the authentication code could not easily coexist in the same process space as the Netscape server. Given that source code for neither system was available, and that it seemed likely there might be more vendor-supplied APIs or similar complications in our future, it seemed wisest to remove all authentication and authorization decisions from the Netscape server, and place them in an external process.

BigDog and the web server communicate with a simple, easily-extended protocol. BigDog has its own user databases and web page access-control lists, and maintains an external session database. To get the Netscape web server to work with BigDog, we only had to write two simple NSAPI functions that behave like RPC stubs, passing the parameters of requests to BigDog and

<sup>5</sup>The quality and volume of NSAPI documentation has improved tremendously in the last year.

<sup>6</sup>Given the torrent of books on all aspects of the web, we're surprised that no one has published a book on writing NSAPI modules.

instructing the web server how to handle BigDog's responses. Of course, we also had to build user databases and access-control lists for our web site.

The decision to make BigDog a standalone application has paid off. Getting BigDog working with a new or different web server is far easier than it would have been if BigDog's services had been implemented entirely in any web server's native extension mechanism, and we can make changes to the authentication and authorization semantics without affecting the web server itself. We've also discovered that for purposes of testing and administration, it's useful to have several web servers using the same BigDog: they can share the same set of authentication and authorization databases, and the session concept makes even more sense when applied not just to one web server but to a group of them.

## 5 BigDog Administration

There are only three things to administer in managing BigDog: user databases, ACLs, and the BigDog process itself.

### 5.1 User Databases

BigDog user databases are stored in a binary Btree database. We chose the database format to optimize lookup time, simplify approximate searches, and leave room in the future for real-time updates.

A BigDog user record contains the following fields:

**user** ■■■■■ any user@domain.name style address.

**password** ■ “multi-use” password, which can be null (indicating that the user does not need a password to gain password-level access to the site).

**ID number** a unique number for each user; typically, this is the primary key of a table in a separate relational database we maintain on all users.

**group ID number** the primary key of a database containing information on logical groupings of user IDs.

**common name** the full personal name of the user.

**class** a descriptive name of the type of user ID.

**idle timeout** the maximum amount of time to allow between user interactions in a session before requiring reauthentication.

**authentication timeout** the maximum amount of time between user authentications.

{//foo.ms.com/cgi-bin/foo.pl}	*@ms.com	SecurID-Auth	hostname=ms.com
{//foo.ms.com/index.html}	bf@ms.com	5	serf=yes,peon=no
{//ms.com/index.html}	mtk@columbia.edu	45	peon=yes,phat=yes

Figure 2: Some sample ACLs

**home domain** the name of the user's home domain.

For doing batch-style user database updates, ■ compiler translates newline-separated, colon-delimited ASCII files (much like Unix's `/etc/passwd`) into the binary format used by BigDog. A C/C++ API for altering the database on the fly is also available. The password-setting code performs a number of checks to prevent users and administrators from selecting easily guessed passwords.

## 5.2 Access Control Lists

BigDog understands two access control list formats, a binary hash file, and an ASCII text file. Unwildcarded ACLs can be placed in either, but only the ASCII file can contain wildcarded ACLs (recall that ACLs can be wildcarded on user name or URL). We have written a compiler that translates ACLs without wildcards from ASCII into our hash database format.

Figure 2 shows some sample ACLs. The first field, contained in '`{}`' is the path of the URL this rule applies to. The second field is the user (or users, if it contains a wildcard character) this ACL applies to. The next field is the minimum authentication level required for access to the URL by the listed user. Note that authentication levels can be either numeric or alphabetic. The final field is optional and contains a list of comma-separated name/value pairs to be placed in the process environment by the web server before servicing the URL.

## 5.3 Adding BigDog to ■ web site

It's very easy to add configure a running web site to use. If the web site is running a Netscape web server, a total of three lines have to be added to all the server configuration files: one line to load the new server functions, and two new server functions which pass authentication and document request information from the web server to BigDog and parse and process BigDog's response.

We have not written BigDog stubs for other web servers, but most recent web servers provide extension mechanisms similar in intent to the NSAPI. It would not be hard to port our NSAPI code to some other server, especially for web servers that come with source code.

## 6 Related Work

### 6.1 Open Market's OM-Axcess Server

Although we have no direct experience with it, Open Market's OM-Axcess appears to be similar in intent to BigDog.[3] It is a central authentication and authorization server for web servers, provides sessioning facilities, and has an API for altering and expanding its authentication and authorization routines. Unlike BigDog, it encodes an opaque session identifier in the URL. It also relies on the session identifier to transmit all user capabilities. Once ■ browser has an OM-Axcess session identifier, no further HTTP authentications are performed.

One of our baseline assumptions was that the environment BigDog and our web servers run in is relatively secure—that we did not have to worry about Trojan horse BigDogs or counterfeit web servers attempting to use BigDog.<sup>7</sup> OM-Axcess, on the other hand, mutually authenticates itself to the web server using public key encryption techniques.

### 6.2 Other sessioning schemes

Techniques for session identification are common on the web today, and existed long before BigDog did. Many of these schemes encode the session identifier in the document URL. Pathfinder (<http://pathfinder.com>) may be the best-known site employing such a scheme. This scheme has some nice features. Most importantly, the web browser does not have to support cookies. However, this scheme creates problems, too. It makes it possible to "shoulder surf" session identifiers.<sup>8</sup> Bookmarking such URLs can cause strange behavior if the session has expired. Finally, session identifiers embedded in URLs can look strange and be unpronounceable (we frequently pass URLs to friends and colleagues over the phone. Can you imagine telling someone to "visit [http://foo.bar.com/@@\\$qAaa%ss9V@@@UK/equities/index.html](http://foo.bar.com/@@$qAaa%ss9V@@@UK/equities/index.html)"?).

<sup>7</sup>We do regular and exhaustive checksumming on all web server file systems to help insure this.

<sup>8</sup>Now that web browsers can be configured to request confirmation before accepting a cookie, this problem is also present to a lesser degree in cookie-based schemes ■ well

There are many web packages that implements authentication techniques similar to one or more of the methods BigDog implements. Many systems allow the user to define new authentication schemes. We know of no others that explicitly attempt to unite many authentication techniques in a coherent framework, and that are web server-neutral.

## 7 Planned Work

### 7.1 Real-time update

We designed BigDog with a real-time publish/subscribe data update system in mind for its access-control lists and user databases. We plan to move these databases into a relational database system, and to implement triggers which publish the changes into a “data cloud” in real time. Whenever a BigDog comes up, it will subscribe to updates in its access control and user databases from the real-time data cloud.

### 7.2 Replication

High availability and reliability are essential to providing quality services to our clients. Because a BigDog breakdown would disable our client web services, we built an open-ended replication system into it. All BigDog servers will subscribe to real-time updates of the user and ACL databases, and “packs” of BigDog servers will be able to asynchronously update each others’ session databases, without affecting the performance of their authentication and authorization functions.

### 7.3 Embedded Perl Interpreter

Adding an interpreter to BigDog is clearly the next step in flexibility and extensibility. We can already envision situations where we will want to inject new access control or authorization logic into a running BigDog, or to be able to override default behaviors on the fly, or issue very specific queries. Recent versions of perl are relatively easy to embed in another application, and we plan to expose most of BigDog’s internals as perl primitives, so it will be possible to implement substantial new functionality directly in perl. We already use a perl interpreter that only accepts digitally signed perl code, and we will place these same checks in the interpreter we embed in BigDog.

## Conclusion

“Electronic Commerce” has almost as many definitions as it does practitioners. For us, it means securely carrying out any of a large set of business-related communications with our clients through our web site. The flexibility BigDog gives us in controlling access to that web site is a key enabler for our electronic commerce efforts.

## References

- [1] David M. Kristol and Lou Montulli. *Draft-ietf-http-state-mgmt-03: Http state management mechanism*, 1996.
- [2] *Netscape Commerce Server Programmer’s Guide*, chapter 2: Netscape API functions. Netscape Communications Corporation, 1995.
- [3] Open Market, Inc. *OM-Axcess<sup>TM</sup>: Technical Overview*, May 1996.

# Financial EDI Over the Internet, Case Study II: The Bank of America and Lawrence Livermore National Laboratory Pilot

Arie Segev, Jaana Porra, and Malu Roldan

segev@haas.berkeley.edu, porra@haas.berkeley.edu, roldan@haas.berkeley.edu

*The Fisher Center for Information Technology & Management*

*Haas School of Business*

*University of California, Berkeley*

## Abstract

Bank of America Corporation (BofA) has used Electronic Data Interchange (EDI) to transmit financial transactions between itself and its customers for years. Until recently, however, BofA has used direct private lines or third party value added networks (VANs) as the carrier of the EDI data. In 1994, BofA initiated a pilot project with its customer Lawrence Livermore National Laboratory (LLNL) to investigate whether the Internet could be used for secure, reliable, and fast financial EDI (FEDI) transactions. This follow up study is the second case study on BofA's EDI strategy conducted by the Fisher Center for Information Technology and Management at the University of California, Berkeley. The first case study was completed in August 1995 (Segev, et.al., 1995). In the first case study, the current practices in EDI, FEDI, and related technologies were reviewed and documented. In this second case study, the results of the pilot project, which ended June 30th, 1996, are presented and possible implications are discussed. According to the results, the Internet is a viable alternative carrier for critical or sensitive business transactions. It is suggested that the transition from the traditional EDI to EDI over the Internet has strategic implications.

## 1. Introduction

Electronic Data Interchange or EDI is a synonym for standardized inter-organizational data exchange (Kalakota, et.al., 1996). It has been used for more than 20 years in automating the interchange of business transactions between trading partners, suppliers, and customers in order to create a paperless flow of administrative, prepurchasing, purchasing, shipping, receiving, warehouse, customs, billing and payment information (Sokol, 1995). Forrester Research has estimated that today 120,000 of 2 million companies are EDI capable (Smith Bers, 1996). This means that only 5 percent of the companies that could be using EDI are

actually doing so. For large corporations, who provide EDI based services, this means that 95 percent of the market is still to be harnessed.

Bank of America Corporation (BofA) has used Electronic Data Interchange (EDI) to transmit financial transactions between itself and its customers for years. Until recently, however, BofA has used direct private lines or third party value added networks (VANs) as the carrier of the EDI data. In 1994, BofA initiated a pilot project with its customer Lawrence Livermore National Laboratory (LLNL) to investigate whether the Internet could be used for secure, reliable, and fast financial EDI (FEDI) transactions. This follow up study is the second case study on BofA's EDI strategy conducted by the Fisher Center for Information Technology and Management at the University of California, Berkeley. The first case study was completed in August 1995 (Segev, et.al., 1995). In the first case study, the current practices in EDI, FEDI, and related technologies were reviewed and documented. In this second case study, the results of the pilot project, which ended June 30th, 1996, are presented and possible implications are discussed. According to the results, the Internet is a viable alternative carrier for critical or sensitive business transactions. It is suggested that the transition from the traditional EDI to EDI over the Internet has strategic implications.

## 2. EDI

Electronic Data Interchange (EDI) refers to business-to-business exchange of data (Sokol, 1995). It requires evaluating strategic and operational issues such as EDI related organizational change, service and development responsibilities, development of EDI based services, and their marketing. In the widest sense, EDI refers to all policies, procedures, and technologies related to the exchange of EDI information between two corporations. Defined this way, EDI includes the processes of procurement (handling of requests for quotes, price quotes, purchase orders, acknowledgments and invoices).

This means that implementing EDI requires not only developing or purchasing the EDI software and hardware, but also organizational change through Business Process Reengineering (BPR).

An EDI relationship starts with an agreement between a company and its trading partner (Segev, et.al., 1995). The agreement consists of technical decisions concerning the EDI standard, types of information exchanged (e.g., purchase orders, payments, etc.), and the exact format of each message exchanged. Other important decisions concern both technical and business aspects of the EDI relationship. These include agreements about the interbusiness procedures related to EDI such as why and when messages are sent; how are messages responded to; and what are the procedures for handling exceptions.

In addition to the format and procedural EDI decisions, an EDI relationship requires agreements on the degree of automation at both ends of the relationship. One of the main purposes for using EDI is the reduction in manual processing of paper based transactions. Reducing manual, paper based processing, in turn, leads to cost reduction and faster service to the customers. This means that the benefits of EDI are directly related to the degree of automation.

The higher the number of the human interventions, the more time and paper the process requires. Processing paper by humans is more costly and time consuming than using automated processes. Reduced processing speed, in turn, translates to lower quality customer service. Interorganizational EDI thus may be thought of as an interorganizational, fully automated assembly line (Figure 1).

Since EDI refers to a standard not to a business process, it is often presented not as an interorganizational information system, but in a purely technical way. In essence, EDI defines the *standard* format for exchanged *messages*. In this context, a *message* refers to a document such as a purchase order, an invoice, a payment, or any other formatted business-to-business, administrative, purchase, or sales transaction. *Standard* refers to one of the several international (e.g., EDIFACT), national (e.g., ASC X12 in the United States), or industry wide (e.g., National Wholesale Druggists Association (NWDA); and Uniform Communications Council (UCC)) agreements which define the data items on the EDI message (the number, type, order, and precise format for the fields of the document). Today, EDI standards initiatives take place at several fronts (Sokol, 1995).

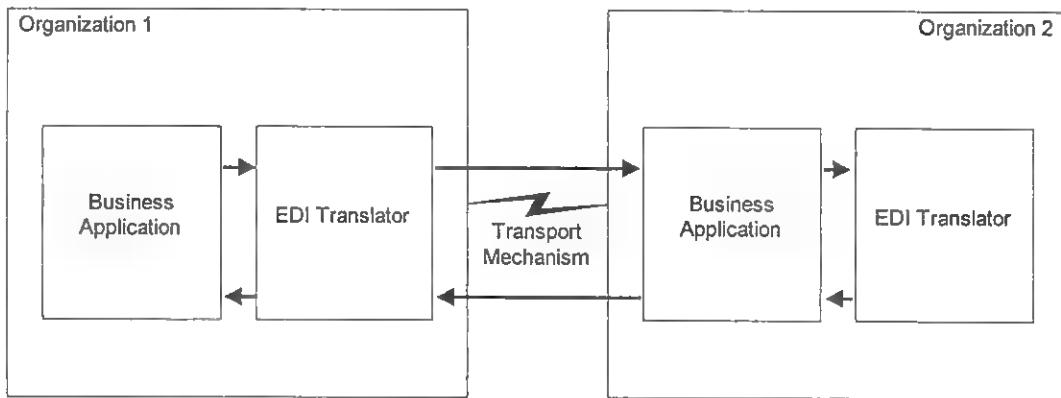


Figure 1. Automated Interorganizational Data Flow with EDI.

Today, it is possible to choose from a variety of EDI standards such as the ones mentioned above. Moreover, inside each standard, it is often possible to choose between several profiles (characteristics added by selected optional features of the standard). The purpose of standardization is to reach a general agreement on which current and future EDI users will commit to. A single standard, with no optional aspects, would guarantee that any two corporations were able to exchange EDI messages without negotiation. Many EDI

standards, each with several options, create an array of possible formats for an EDI transaction. The advantage of using a standard, is in the predefined public agreement concerning the transaction format. The downside of the current EDI standardization situation is the number of possible standards and options which may make using an EDI standard unattractive. Luckily, the situation is not as serious as it may seem. Of the EDI standards, the two most likely candidates for a wide acceptance, are the ASC X12 in the United States, and EDIFACT internationally. Thus, using either of the two will increase

the likelihood for successful EDI relationships at home and abroad.

In the past, the initial cost of integrating EDI into the corporate information system has been one of the main issues preventing EDI's success. Traditionally, these EDI implementation costs have included the cost of purchasing or developing the EDI translator, the cost of integrating the EDI translator with the corporate information system, and the cost of establishing a transport mechanism (often provided by a telecommunications carrier). The first two cost categories consist of purchasing the hardware, software, and programming skills. Choosing the transport mechanism includes choosing between private networks, Value Added Network providers (VANs), and more recently, the Internet. Supporting the production system has been another, sometimes initially overlooked, contributor to the high cost of EDI. Once the software is designed and implemented, the cost of handling paper based transactions is reduced. What sometimes escapes the corporate cost benefit analysts, is the fact that the paper processing cost is partially replaced by the increased need for maintaining and operating the computer based EDI service.

### 3. FEDI and FEDI over the Internet

Bank of America Corporation (BofA) is one of the largest banking companies in the United States with assets of more than \$227 billion (Segev, et.al., 1995). As a diversified global financial services institution, BofA provides banking products and services to individuals, businesses, government agencies, and other financial institutions in the United States, and in 36 other countries. Today, BofA supports all major electronic payment options, including FedWire funds transfer service; automated clearinghouse transfers or ACH; Clearinghouse Interbank Payments System or CHIPS service; and Society for Worldwide Interbank Financial Telecommunications or SWIFT funds transfer service. These transactions do not use EDI or FEDI. Rather, FEDI comprises the electronic transmission of payment and remittance instructions and acknowledgements between a payer, payee, and their respective banks (Figure 2). The BofA pilot project concentrates on FEDI between BofA and its customer, LLNL. Like most all other corporations engaged in EDI (Messmer, 1996), BofA has been using either direct private lines or third party vendors called value added network service providers (VANs) as the EDI network carrier. The objective of the BofA pilot was to evaluate the Internet as an alternative.

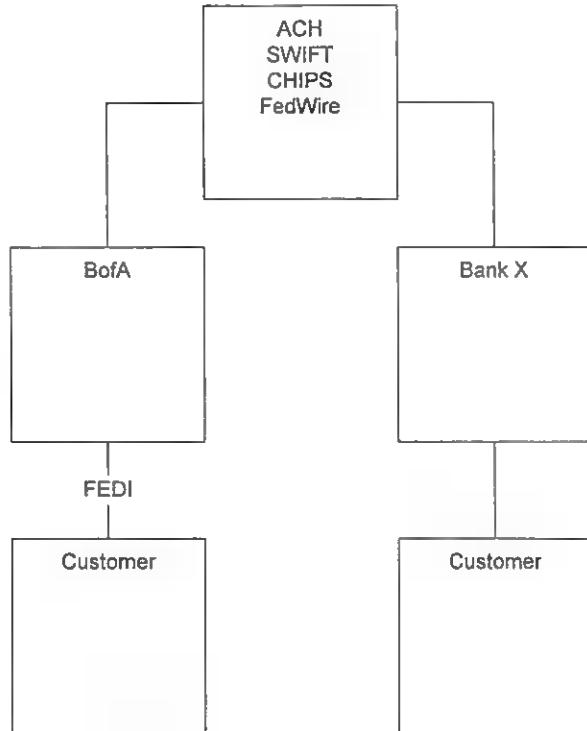


Figure 2. Financial transactions and FEDI.

In the case of BofA, the purpose of FEDI based customer service is to offer a set of the most widely used EDI standards to customers, rather than to advocate a single EDI standard. Another goal for the FEDI based customer service, is to integrate the EDI based solution with BofA's information systems without requiring comprehensive automation on the customer's part. The same desire to meet the customer's needs is apparent in BofA's response to the transport mechanism options. BofA will offer multiple transport mechanisms to serve a variety of customer needs. The case at BofA shows that EDI is rapidly overcoming several of its limitations. Today's flexible and cost effective EDI solutions can be used between large corporations, and increasingly in business transactions with small enterprises. Large corporations, such as BofA, are thinking ahead for their customers when selecting EDI solutions that alleviate the problems of incompatibility by offering multiple access methods.

The two main reasons for considering the Internet for FEDI transactions at BofA, are cost savings and new market potential. Both private lines and VANs charge based on the transaction volume while the Internet providers charge a flat fee (Jetter, in Smith Bers, 1996). Particularly, with high transaction volumes, the start up costs, fixed costs, and usage costs of the Internet are

significantly lower than those of private networks or VANs (Segev, et.al., 1995). The other main reason for considering the Internet as the FEDI transaction carrier is derived from two market segments: (1) the 95% potential, consisting of large EDI capable corporations which are not currently using EDI; and (2) the fast growing number of Internet users, who are mainly small businesses and individuals. Today, it is estimated that the Internet consists of more than 30 million computers world wide. The latest easy to use multimedia interface to the services of the Internet functionality, the World Wide Web (WWW), is said to be doubling in the number of users every 53 days (Pant, et.al., 1996). Even when measured merely by its size, the Internet and the WWW provide a compelling case for new business strategies. The modern Internet, and particularly the WWW, are the center of electronic commerce of which EDI is an integral part. A third, often overlooked reason to consider the Internet for EDI and FEDI transactions is that the Internet is already used in most organizations. Using Internet based electronic mail as the messaging system draws from familiar and available resources.

#### **4. Internet Related Concerns – Security, Reliability and Speed**

The Internet security concern has been supported by widely publicized compromises of Internet security (Bhimani, 1996). Over the decades, reports of actual Internet security problems have been confused with reports of security problems which were not caused by the Internet, but the computer systems attached to it (Hutt, et.al., 1995). The lack of real world examples of consistently delivering critical transactions over the Internet, keeps many corporations as spectators in the Internet explosion. However, the situation is rapidly changing as better technologies are being developed, and web-based transaction systems are becoming a high priority initiative for corporations in shipping, banking, airline, and retail industries.

Two other central concerns, (1) reliability, and (2) speed of transactions over the Internet, are founded on the original design principles of the Internet network. There is no centralized management and no centralized routing of the transactions. Instead, the management of the Internet, and the routing of the messages, are distributed. This means that no one organization or instance is responsible for carrying the transaction from the sending computer to the receiving computer. Moreover, the routing of a message is dynamic. There is no way of predicting or controlling the route the data will take from the transmitting computer to the destination. Corporate

network service users are accustomed to a third party, who guarantees a level of service, and reliable, fast delivery. A common perception is that the Internet architecture has proven to be adequate in advertising and communications which are the two main business uses of the Internet today (Stipe, 1996). However, the concerns about lost data and prolonged transmission times will require addressing in the corporate transaction setting. Today's commercial Internet, and particularly the WWW, have corporations as users. In a corporate environment, the fastest transaction processing systems are based on executing real time transactions (the message is sent from the sending application to the receiving application at the moment of initiation of the transaction). Security, reliability, and speed are now being measured according to the corporate standards comparable to those used in private networks and by third party VANs.

#### **5. The FEDI Pilot**

The previous sections provided a general overview of EDI and FEDI, the Internet, their organizational impacts, and BofA's use of EDI. This section describes the parameters and operational results of the FEDI pilot project between BofA and LLNL. Section five presents conclusions and an examination of the above issues in the context of the pilot.

As discussed above, the Internet presents a seemingly attractive alternative to BofANet and VANS. It is widely available and reputably inexpensive when compared to VANs. Since the bank already has an Internet connection for other applications, e.g. electronic mail, the incremental cost required to carry EDI traffic over the Internet is minimal. The bank views the flat-fee, volume-independent and time-of-day independent pricing structure of the Internet as one major benefit over other types of networks, like VANs. Because of the large volume of transactions, the bank expects to achieve tangible savings, over the long term, by either redirecting EDI traffic from other channels, generating new customers from current Internet users, or both.

Thus, in late 1994, individuals at BofA were searching for a client partner to start a FEDI pilot project. Because of general concerns over Internet security, particularly for transmissions involving financial transactions, BofA expected that it would have a difficult time finding a willing partner. At about the same time, individuals at LLNL became interested in building the capability for FEDI over the Internet, as part of a planned overhaul of its accounts payable system. At LLNL, this was seen as a next step in its continuing leadership with innovation in

the practice of EDI. Quite fortuitously, representatives from both organizations made contact at several CommerceNet and Special Interest Group meetings in the Bay area, and decided to pursue a possible pilot for conducting FEDI over the Internet.

After high level talks, both organizations decided on a limited pilot, requiring minimal investment from the parties involved. The pilot involved participation from several departments at both organizations – most strongly from BofA's Global Payments Services and Interactive Banking groups, and LLNL's Accounts Payable and Electronic Commerce departments. The key objective of the pilot was to demonstrate that it was possible to achieve secure and reliable transmission of sensitive data like payment instructions over the Internet. The pilot was meant to dispel negative perceptions of the Internet held by the general public and the business community. The pilot would be especially valuable in addressing the concerns in the Banking Industry over sending critical financial data over the distributed public networks of the Internet. As discussed above, these perceptions centered primarily on two concerns: security -- that transactions over the Internet were easily intercepted and tampered with, and reliability and speed -- that transactions tended to get lost and/or delayed as they were passed from node to node over the Internet.

## 5.1. Chronology of the Pilot

The FEDI pilot included two phases. The first phase limited the exposure of both LLNL and BofA to the risk of financial losses due to compromised payment instructions. Success with this limited test led to the relaxation of the limits during the second phase of the pilot. The second phase also included volume testing to simulate any problems that might result from a full-scale implementation. All throughout the seven months of the pilot, BofA project participants were also involved in designing a full-production platform to support FEDI and similar applications.

FEDI pilot participants from LLNL and BofA spent most of the first half of 1995 designing, coding and testing the FEDI system. At BofA, the pilot required development of a gateway system for encrypting and decrypting EDI documents that it would transmit to its existing EDI and ACH systems (running on its ECS server). This gateway was located inside the bank's firewall. Part of an existing Unix machine on that platform was allocated to the project. The main software development work involved installing PEM/MIME on this Unix machine and automating the file handoffs between various BofA

systems. At LLNL, the pilot was part of an effort to improve the accounts payable system. Since the Lab already had an existing PEM/MIME server and EDI translators, the main development work involved automating the handoffs between various LLNL systems and creating reports used to monitor the progress of transactions throughout the system. Additionally, during the testing phase, LLNL's implementation of a new version of their A/P database necessitated further development work to make the translators and handoffs work with the new database.

In August 1995, the first payments instructions were successfully sent from LLNL to BofA, resulting in payments to selected vendors. This started the first phase of the pilot project. To minimize risks, the pilot was limited to five vendors and a cap of \$10,000 was established for payments to a single vendor per day. The process included vigilant monitoring by key pilot representatives of both organizations. E-mail messages were used to track the progress of FEDI documents through the system. Any anomalies were quickly identified and resolved. The key pilot representatives were in touch with each other via phone and beeper systems. Weekly meetings were used to track progress with resolving recurring problems. Because of this vigilance, all payments were made within the contract time frame – that is, all payments were received by each vendor's bank within two days of the generation of payment instructions from LLNL.

The first seven months of the pilot convinced both organizations of the viability of sending payment instructions over the Internet using a PEM/MIME-based security system. Measures of security, reliability, and speed -- described in section 4.4 -- provided a measure of confidence that the instructions were not lost in the Internet or tampered with. Both organizations were, at that point, ready to step up their levels of exposure. LLNL added new vendors to the system and increased the cap on payments to \$100,000 for each vendor per day. Additionally, LLNL started a program to use the same process to provide Travel and Entertainment reimbursements to their employees. As of July 1996, LLNL had approximately 12 vendors on the system and about 100 employees signed up to receive Travel and Entertainment reimbursements using the FEDI system. Lastly, to test the robustness of the system to full implementation, the two organizations conducted volume testing. This involved tracking the time it took to process transmissions that contained up to 1000 payment instructions. The results of these tests are reported in section 4.5.

Even as they were going through the pilot, the BofA project group knew that they needed to design a production system to eventually replace the pilot system.

They anticipated that the current system would be impractical for meeting the demand for supporting secure transactions over the Internet -- particularly if the pilot proved successful and the numerous parties interested in the pilot signed up for the service. Through discussions among representatives of the bank's security, marketing and information systems areas, as well as with key hardware vendors, the project group designed both an interim system and a full production system. At the same time, as interest in the FEDI project and the Internet grew outside and within the bank, senior management started to become amenable to possibly investing in better FEDI platforms. The project group obtained funding for an interim production system and as of July 1996, were waiting to hear if the full production system would be funded.

BofA has obtained the hardware and software to move their FEDI operations to an interim platform. This was the first step in moving the FEDI platform from a research to a production environment. The interim system involves software and hardware improvements. The new platform uses Trusted Information Systems' (TIS) MOSS software instead of PEM/MIME. MOSS - MIME Object Security Services is a proposed standard that extends PEM capabilities by supporting e-mail messages with arbitrary contents (including video, voice, images), improving functionality and expanding trust and naming rules. The hardware configuration of BofA's interim system involves three Unix workstations. Two of the workstations will be running MOSS in parallel and will be located in the bank's production environment in San Francisco. The third workstation will be used for development purposes and will stay in the Concord facility. BofA plans to add other EDI packages (e.g. Premenos' Templar) as new customers request them. The proposed production system currently under review at BofA, brings security to a higher level by not only using software but also hardware to achieve it. Aside from being more secure, a hardware-based system also speeds up encryption and decryption processes. The key to the final system is a hardware level security server. This server will make sure that no sensitive data -- e.g. payment instructions -- is ever visible as clear text during transport over the Internet or participating organizations' internal networks.

## 5.2. The Pilot System

The following description is organized around three aspects of the pilot system -- the exchange of documents, the achievement of security, and the monitoring of system performance. All three are key to making the case for the viability of FEDI over the Internet. In general, the system involved the exchange of EDI documents containing payment instructions and acknowledgements. To achieve security, the documents were processed through servers running PEM/MIME at entry into and exit from each organization's existing network of EDI systems. A system of e-mail and human monitoring tracked messages through the system -- insuring that payments were completed accurately, and collecting data to assess system performance.

**FEDI Document Exchange.** The process starts when LLNL extracts a batch of payment instructions (EDI document 820) from its Accounts Payable database. After extraction, LLNL's PEM server processes the batch of 820s according to the security measures described below. LLNL then transmits the secured batch of 820s to BofA via the Internet. Upon receipt of the LLNL message, BofA's PEM server processes the batch to confirm its authenticity and decrypt its message. The clear text message, containing a batch of 820s, is then transmitted through a dedicated line to BofA's EDI server (ECS). BofA's ECS server then confirms that the data was received, that it is in the right format, and contains the correct information. The ECS server then sends back a functional acknowledgment (EDI document 997) to the BofA PEM server for security processing. The PEM server, in turn, transmits the secured 997 over the Internet. Upon receiving the message, LLNL's PEM server authenticates and decrypts this 997. It then matches up the information in this 997 with its record of the information contained in the original 820 document that it sent to BofA.

Meanwhile, BofA's ECS server completes processing of the batch of 820 documents, sending payment instructions to a payments server -- e.g. ACH for electronic funds transfer, or a check processing system. After handing off the instructions to a payments server, the ECS system sends an Application Advice (EDI document 824), back to BofA's PEM server. The 824, containing information on payment acceptances and rejections, is secured and transmitted to LLNL over the Internet. LLNL authenticates and decrypts the message then matches it up with the information set out in the original payment instructions (820) and with BofA's acknowledgment (997). If all the information matches

up, LLNL sends a secured 997 functional acknowledgment to BofA through the Internet. On average, if problem-free, this whole process takes 11 minutes (Average of logged time for problem-free transmissions computed from BofA's Daily Tracking Log, July 12, 1996).

The agreement between LLNL and BofA requires that a vendor bank receive payment within 2 days of LLNL sending an 820 to BofA. Since the entire FEDI transaction takes only an average of 11 minutes, both organizations have ample time to research and resolve any problems related to the transmission of a given payment between BofA and LLNL. Problems could also occur if the vendor's bank rejects a payment request for a variety of reasons – e.g. insufficient funds, vendor has changed banks. Should a vendor's bank eventually reject a payment instruction, BofA receives a financial return notice from the vendor bank (EDI document 827). As with the other documents, this 827 is secured by BofA's PEM server and sent to LLNL over the Internet. Personnel at LLNL's Accounts Payable department then conduct the research to determine why the payment instruction was rejected then take the necessary steps to resolve the problem.

**Security Measures.** A system of multiple acknowledgements, information matching at LLNL's EDI server, and encryption and signing of all e-mail transmissions containing EDI documents formed the basis for addressing the minimum security requirements discussed in section 3.2 – confidentiality, authentication, data integrity, nonrepudiation, and selective application of services (Bhimani, 1996; IETF-EDI Working Group, 1993). Confidentiality was achieved using encryption, while a system of digital signatures, encryption and one-way hash functions helped achieve authentication, data integrity and nonrepudiation. Lastly, selective application of services was achieved by transmitting any clear text only through dedicated network lines. Any text that went through shared networks like the Internet was encrypted and digitally signed as PEM/MIME documents.

At the start of the pilot, the two organizations had to exchange public keys to be used in RSA encryption and decryption procedures. Ideally, a third party, called a Certifying Authority, would vouch for the authenticity of these public keys. Since the two parties had an ongoing relationship based on trust, and due to the lack of Certifying Authorities, the two organizations essentially certified themselves as bound by their respective public keys. Both LLNL and BofA also had to agree upon the algorithms to be used for security procedures. For the

purpose of the pilot, the algorithms were part of the PEM/MIME software acquired from TIS – RSA's PK1 for asymmetric public key encryption, DES for symmetric encryption, and RSA's MD5 for the calculation of one-way hash values. Additionally, the pilot participants had to agree on appropriate lengths for the encryption keys. As with any other encryption process, an important determinant of the strength of the security associated with this standard depends on the length of these keys.

In essence, the security procedures consisted of three major steps. The first step was used to insure the confidentiality of the EDI document through encryption. Encryption makes it difficult for unauthorized individuals to read the document while it is in transit to the receiver's system. The second step, use of a one-way hash procedure, insures the integrity of the document contents. Using an agreed upon algorithm, the sender calculates a hash value for the EDI document and sends it to the receiver in encrypted form. Using the same algorithm, the receiver can calculate a hash value for the EDI document it receives. If the hash values match, the receiver can have some level of assurance that the document was not tampered with. In the last step, the sending organization guarantees its identity by sending a digital signature. The digital signature consists of the one-way hash value encrypted by the sender's private key. The receiver can then use the sender's public key to decrypt this hash value and match it to the hash value it calculates.

The security procedures apply every time an EDI document comes to a PEM server – e.g. when an 820 is sent by LLNL Accounts Payable system to LLNL's PEM server or when BofA's ECS server sends an 824 to BofA's PEM server. The sender's PEM server first calculates a one-way hash value for the document using the MD5 algorithm. It then generates a symmetric key and uses the DES algorithm to encrypt the EDI document. Next, it encrypts the symmetric key with the receiver's public key, and encrypts the one-way hash value with the sender's private key. The EDI document, symmetric key, and one-way hash value – all in encrypted form -- are then bundled in a MIME envelope and sent over the Internet to the receiver's PEM server. Upon receipt of the message, the receiving organization's PEM server first authenticates the sender's identity by decrypting the one-way hash value using the sender's public key. It then decrypts the symmetric key using the receiver's private key. Next, the decrypted symmetric key is used to decrypt the EDI document. This document is then sent on to the receiver's EDI server for processing.

The PEM/MIME security procedures were set up to meet minimum security requirements, at high speed, and with simple key management. They involve a combination of symmetric key encryption (DES) and asymmetric public-key encryption (RSA). This system speeds up the encryption/decryption process as well as simplifies key management. Under this system, the faster DES algorithm is used to encrypt relatively large documents like the FEDI documents. The key used for the DES encryption, a relatively smaller document, is then encrypted using the sending organization's private key. Key management is simplified so that the two parties need to exchange only their public keys. There is no need to exchange symmetric DES keys since a new, random DES symmetric key is generated for each transmission. This symmetric key is encrypted with the receiver's public key and sent along with the PEM/MIME message. An additional benefit of this system is that the DES keys are linked to only one EDI transmission at a time. An individual who could figure out a given DES key could only pose a threat to that one transmission. In contrast, if the two organizations used the convention of agreeing on a single DES key to be used for a range of transmissions, any individual who could figure out this agreed upon DES key could potentially decrypt a large number of transmissions (IETF-EDI Working Group, 1993.).

**Monitoring the System.** The pilot system required multiple confirmations of the information transmitted at key steps in the process (using 997 EDI documents). This system of acknowledgements provided one method of establishing the reliability and security of the process. Additionally, key contacts at each organization monitored each phase of the process and would telephone his/her counterpart if an error or omission occurred in the process. BofA summarized the movement of each and every transmission between BofA and LLNL through this process for the seven month duration of the pilot in a Daily Tracking Log (July 12, 1996). At each point where a time reading is taken, an email message is generated by the LLNL EDI server and sent to key personnel at BofA and LLNL. In this way, several individuals kept vigilant watch over every transaction and were able to quickly determine and react to any errors or delays.

Security of the network was monitored using standard procedures at both organizations. The groups managing the firewalls of each organization conducted their own tests and monitoring procedures to keep track of both legal and illegal access attempts to each organization's network. These procedures were part of the general network security protocols at each organization and were

not done especially for the FEDI project. None of the informants reported on the use of procedures to trace the transit of message packets through the Internet.

Based on the data from BofA's daily tracking log, we generated a summary of the problems encountered throughout the pilot (Table A). As the table shows, almost 50% of the problems stemmed from systems going down or off-line. This figure reflects the difficulty of coordinating the separately managed systems that were all part of the FEDI pilot process. This condition was magnified because BofA's PEM server was part of the firewall operation that resides in an R&D location – as opposed to a financial production environment. Thus, BofA's PEM server was at times accessible to individuals who were unaware of the impact on the FEDI pilot of any processes they might run on the PEM server. Furthermore, both BofA and LLNL went through several software problems and upgrades over the course of the pilot – resulting in further problems with compatibilities between the systems in the network. Much of the early work done by the project team involved solving these incompatibilities and instituting procedures to coordinate all the different systems involved in the pilot process. Table A shows how most of these problems were resolved by January 1996. It should be noted though that despite these problems, all payments were completed within agreed upon timeframes – 2 days from the time that LLNL sent a payment instruction to BofA.

Aside from tracking problems, both BofA and LLNL monitored the performance of the FEDI process with the use of a variety of measures. These measures may be grouped according to the top concerns related to FEDI over the Internet—security, reliability and speed.

**Security.** Security measures were used to determine that there were no attempts to access the network or tamper with messages sent between the two organizations. These measures included attempts to break into the system as well as a system of cross-checks and acknowledgements to insure that the data received at one end of a transaction, matched the data sent from the other end. Security measures included the following:

- LLNL had their internal security group run a "white hat" test of the LLNL FEDI network. The network did well and got even better with the implementation of recommendations of the "white hat" team. Additionally, LLNL staff checked the pattern of log-ins to their FEDI system periodically to check for patterns that would suggest security breaches – none were observed.

Table A. Problem Summary BofA/LLNL FEDI Pilot (Based on BofA's daily tracking log)

Type of Error	Aug. 1995	Sept. 1995	Oct. 1995	Nov. 1995	Dec. 1995	Jan. 1996	Feb. 1996	Mar. 1996	% of total problems
Applications, Operating System incompatibilities	2	1			4				17%
Systems going down or off-line	7		6	1	2	1	2	1	49%
Document delivery problems (duplicate, delayed, or lost documents)		1	3	3	2		1		24%
Message problems (truncation)			2						5%
Decryption problems						1	1		5%
A. Total # of Problems for the month	9	2	11	4	8	2	4	1	
B. Total # of EDI transmissions for the month	19	21	22	22	21	23	21	21	
Error Rate for the month = A/B	47%	10%	50%	18%	38%	9%	19%	5%	

- Dollar amounts, payee information, transaction numbers and EDI formats were checked and matched using the series of EDI documents (820-997-824-997-827) described in section 4.0. All of the mismatches generated by this system were explained with reasons other than security breaches. Additionally, as noted above none of these problems resulted in payments sent beyond agreed upon timeframes.

- The group in charge of BofA's firewall tracked all access attempts to the firewall. Additionally, a member of the project team tried, unsuccessfully, to break into the system by sending an unauthorized file to the ECS system. BofA also contracted with an outside vendor to conduct a test of the firewall's penetrability.

**Reliability.** Reliability measures were used to insure that the data sent by one organization matched the data received by the other. Again these measures were taken as part of the series of cross-checks and acknowledgements sent between BofA and LLNL.

- The LLNL EDI server matched up information on the 820, 997, and 824 and generated error messages whenever it detected a mismatch.

- LLNL personnel checked a daily EDI Batch Control report to insure that the batch total sent was equal to the total acknowledged by the bank.
- A monthly bank reconciliation report used to make sure all payments matched payment instructions sent.

**Speed.** Speed measures tracked how long it took to transmit and process the series of EDI documents between the two organizations. The speed of transmission and processing measures were key to addressing the concern that because of the Internet's decentralized and dynamic routing, messages tend to get lost and delayed. The reliability measures included the following:

- The pilot project team tracked how much time it took a transmission to complete the entire FEDI document exchange process described in section 4.3. This time was tracked by noting the amount of time it took for a transmission to reach seven points in the process. At each of these seven points, e-mail messages were sent to key BofA and LLNL personnel to indicate when EDI documents reached key parts of the process. This data were used to track where a problem occurred when a transaction did not go through properly. This data were also used to calculate the time it took for transactions to complete the loop (from the time LLNL sends the

820 to the time it sends the 997 – after it first receives a 997 and an 824 from BofA. This data was tracked throughout the pilot and during volume testing.

### 5.3. Operational Results of the pilot

**Security.** In the assessment of participants from both BofA and LLNL, the results of the pilot provided support for the viability of the secure and timely transmission of sensitive information over the Internet. Most of this support comes from the finding that none of the problems with payment transmissions were due to any breach of the network or tampering with the messages being transmitted. As shown in Table A, most of the problems encountered during the pilot were due to non-recurring software and procedural issues that were resolved. Problems stemming from the situation where BofA's PEM server was a shared, non-production machine are expected to be alleviated when the process moves to the dedicated interim system in the next few months.

**Reliability.** The most striking finding regarding the reliability of the Internet for FEDI transmission is that none of the messages were lost in transit between BofA and LLNL. The reliability measures also showed that, despite delays and problems, information on payment instructions, acknowledgements, and payments remained consistent. As shown in Table A, there were only two occasions out of 170 transmissions (October 1995) when the EDI document received by the bank did not match that sent by LLNL (duplicate header on 10/2 and message truncated on 10/30). In all the rest of the transmissions, no mismatches were reported by the LLNL server among the FEDI documents sent between BofA and LLNL. This suggests to the participants that, given the security measures used and despite its decentralized nature, the Internet has the capability to accurately transmit critical data like payment instructions.

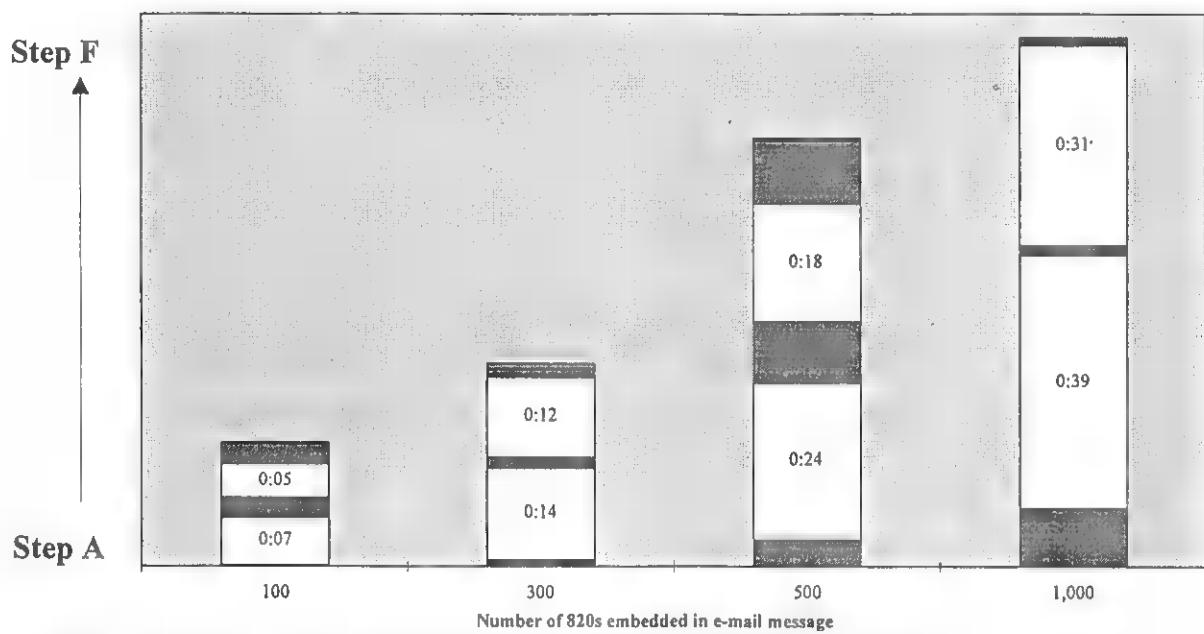
**Speed.** The results of volume testing showed that as the number of payment instructions contained in a message increased, processing time increased (Table B). The total processing time ranged from 11 minutes for messages containing zero to five instructions, to 58 minutes for those containing 1000 instructions. However, this increase can be attributed to the increased time required to process the instructions and not to increases in the time required for transmission of the email message over the Internet. BofA's daily tracking log (July 12, 1996) and Volume Testing log (July 10, 1996) provided the raw data for the findings summarized in Table B. These data were also summarized to delineate time spent on

transmission versus time allocated to processing by BofA servers. The analysis, summarized in Figure 3, shows that most of the time increase resulted from the greater time required to process the transactions at the BofA ECS server (white areas of the figure). There were no systematic or large increases in the message transmission times (gray areas of the figure).

**Table B. Results of Volume Testing – Average Total Processing Time (transmission, decryption/encryption, translation, acknowledgment) for increasing numbers of embedded 820 payment instructions.**

Number of 820 payment instructions in the e-mail message	Average Total Processing Time (from the time LLNL sends 820 to the time LLNL sends the final 997)	Notes
null to 5	11 minutes	N=129 (Average over 7 months of the pilot – includes only problem-free transmissions.)
100	12 minutes	N=8
300	19 minutes	N=6
500	43 minutes	N=7
1000	58 minutes	N=4

**Figure 3. Volume Testing – Summary of Bank of America/LLNL FEDI Pilot Results**



Note: White areas show time required by BofA servers to process EDI documents (Steps B and D).

Gray areas show time spent transmitting messages over the Internet (Steps A, C, E and F).

# of 820 (payment instructions) in the mail message	message containing 820s transmitted from LLNL to BofA over internet	decryption and format checking of 820, generation and encryption of 997	997 transmitted from BofA to LLNL over Internet	BofA's ECS processes payment instructions, generation and encryption of the 824	824 transmitted from BofA to LLNL over the Internet	LLNL system matches information on 820, 997 and 824 and sends a 997 acknowledgement back to BofA over the Internet
Step	A	B	C	D	E	F
100	0:00	0:07	0:03	0:05	0:03	0:00
300	0:01	0:14	0:01	0:12	0:01	0:00
500	0:04	0:24	0:09	0:18	0:09	0:00
1,000	0:09	0:39	0:01	0:31	0:01	0:00

For the pilot participants, the results of the pilot testing showed that many of the problems of security, reliability and timeliness stemmed from problems with their own FEDI systems and not from the use of the Internet as a transmission channel. The pilot participants were satisfied that the system of acknowledgements, cross-checks, and encryption/decryption processes provided a level of performance that is acceptable for sending sensitive information like payment instructions over the Internet.

## 6. Summary

BofA's decision to offer Internet based FEDI services is customer driven. The central question is "when" to begin Internet based services rather than "whether" to offer them. Consequently, the purpose of the *Financial-EDI-over-the-Internet* pilot was to gain confidence concerning the viability of offering Internet based EDI services rather than to sort out the currently available technical solutions for secure Internet transactions. The technical success of the FEDI project was critical. Since BofA is a pioneer in the area, failure to deliver a thorough examination of Internet EDI security, reliability and speed would be costly.

### 6.1 Objectives for the Pilot Project

The challenge for BofA in the *Financial-EDI-over-the-Internet* was twofold. First it was the challenge of publicly promising to exchange actual, secure, financial transactions over the Internet under actual circumstances and with a real customer, Lawrence Livermore National Laboratory, over an extended time period. The second part was to deliver a working implementation. To sort out and choose a viable technical solution amidst the prevailing Internet security debate, is a difficult task. Today, it is effortless to find material and WWW sites discussing and advertising Internet security options. To find out which solutions actually work in the large corporate environment when exchanging critical financial data, was an open challenge. No objective data was readily available.

### 6.2 The Pilot Project ■ ■ Learning Experience

Under the exploratory circumstances, BofA formed an entrepreneurial group, which took up the challenge

while not relinquishing the performance of the members' day-to-day responsibilities. Like so many information system driven ideas, the *Financial-EDI-over-the-Internet* project had to sell itself to senior management. The project group, consisting of two organizations, and more than six business units developed ways of collaborating through large group meetings, use of the Internet, the organizations' respective Intranets, voice mail, and electronic mail. The team used meetings with experts to learn about the issues relating to EDI. Such meetings sometimes consisted of more than 25 people and resulted in interorganizational and interfunctional solutions. The team proved that critical, FEDI transactions over the Internet works. Moreover, the team showed that projects formed around technological novelties are an effective way of testing and incorporating new technologies into business solutions.

## 6.3 Security

Of the areas of information systems security, the *Financial-EDI-over-the-Internet* pilot concentrated on the technical protection of the special security problem, the Internet. The management of information systems security, physical protection, and security concerns related to the corporate network are addressed elsewhere. Of the special Internet security issues, the project directly addressed data modification, repudiation, confidentiality, and data integrity. Internet security issues related to eavesdropping, password sniffing, spoofing, and selective application of services were out of the pilot's scope.

Pilot projects, such as BofA's and LLNL's, are suitable for focused security risk evaluations. Although it is impossible to claim that successful security attacks did not occur, none were encountered during the months of the pilot project. The careful monitoring of the system at both BofA's and LLNL leads to a confidence level of adequate security in the addressed categories. Increased assurance comes from both parties' security departments which evaluated the system and suggested improvements during the project. When evaluating the success of the *Financial-EDI-over-the-Internet* project, it is important to remember that the system and its security testing will only serve as a proof of concept. On this proof, the final production system incorporating these security lessons and others will be designed. Although done under actual circumstances and with actual transactions, the pilot project provided

a relatively controlled environment and intentionally limited risk.

#### 6.4 Reliability and Speed

The other two concerns, reliability and speed, were also adequately addressed at relatively small transaction volumes. Sending EDI transactions over the Internet turned out to be reliable (no messages were lost), and adequately fast. Again, it is important to remember that the *Financial-EDI-over-the-Internet* pilot was not meant to provide absolute proof of reliability and speed at levels of volumes much higher than a 1000 transactions per day. Rather, the purpose of the pilot was to gain confidence in a relatively small scale. After the level of confidence is achieved, the system can be scaled up.

Since the pilot project ended, BofA has been comfortable with the level of security achieved in the *Financial-EDI-over-the-Internet* pilot project. According to George Cheng, the Senior Vice President of Interactive Banking (a newly formed business unit for electronic commerce applications at BofA) it would be foolish to assume that the Internet is absolutely secure. No absolutely secure business solution exists, paper based or computerized. The *Financial-EDI-over-the-Internet* pilot project provided BofA with a learning experience about the issues and risks involved in Internet based business transactions. The pilot served as a proof of concept. Based on the proof BofA is implementing an improved production system.

### 7. Lessons

The purpose of the *Financial-EDI-over-the-Internet* pilot project at BofA is to test the security, reliability, and speed of the Internet in critical business transactions. Despite the technical orientation of the project, it can be viewed as a strategic project for BofA. It has been said that the risks of banking over the Internet are not technical or security related (Ogilve, 1996). Rather, the risks are strategic and competitive. The threat is that banks will be facing competition from unexpected directions. Some predict that in the electronic era, financial transactions over the Internet service can be offered by a software company as well as by a bank (Ogilve, 1996). Others say that banking can be done by personal computers on the Internet as well as by people in the bank branch offices (Williamson, 1996). According to these scenarios,

banks will be reducing the number of branch offices and employees while increasing their services over the Internet. The fear is that unless banks are among the pioneers of secure transaction processing over the Internet, other industries will claim the market. In this light, it can be claimed that the technical components of the *Financial-EDI-over-the-Internet* project have strategic consequences. The lessons of the project can be viewed from the perspective of EDI, cost, the Internet, and the organization.

#### 7.1 EDI Related Lessons

Today, the project group is confident about the future of EDI. As a large corporation, BofA has the opportunity to influence the shape and form EDI will take. As a member of CommerceNet, BofA is participating in the ongoing development of interorganizational standards of which EDI is one. As a large organization, BofA is also an attractive partner in development of more open EDI solutions by software and hardware vendors interested in the area. It is likely, that EDI will evolve toward easier to use solutions as interorganizational standards are implemented in environments such as the WWW and the Internet influenced by corporations such as BofA and its customers. The pilot study has shown that today's EDI solutions already adequately accomplish the task. Moreover, it confirmed that EDI will be a part of the future services provided by BofA on the Internet.

#### 7.2 Cost Issues

While costs savings are one of the most obvious benefits of using the Internet instead of VANs to transmit FEDI documents, experts take a more cautionary approach. One estimate places transport costs as taking only 5% to 15% of the total cost of a VAN services (Adhikari, 1996). Whether this represents a significant cost saving for a company depends on several factors. These include differences in one-time set-up costs, monthly charges, any usage charges, whether or not messages will be transmitted during the VAN's peak or off-peak hours, and how much the company needs the non-transmission services of the VANs. Table C provides a worksheet for thinking about these issues.

Table C. Worksheet for comparing costs between VANs and the Internet.

Types of Charges	Value Added Network	Internet Service Provider
Set-up Costs		
Monthly Rate		
Character Rates Off Peak vs. Peak		
Other Services used by your company		

How to use the worksheet:

1. Determine how much it will cost to initiate and set-up the system using either a VAN or ISP. These costs include start-up fees as well as any hardware or software purchases, programming and customization (e.g. building EDI translators). At BofA, setting up the PEM server, translators and links to the ECS server cost \$75,000.00
2. Determine basic monthly charges you have to pay even if you do not send any data over the network
3. Decide whether you will send the EDI documents during peak or off-peak hours of the VAN. Approximate how many characters you will send over the network monthly and multiply this by the rate that is in effect when you send the documents
4. List the VAN services that you use (e.g. mailboxes, message tracing). Decide whether those services are worth the price differential between the two channels.

### 7.3 Internet Related Organizational Impacts

Although cost was one of the central issues in the initiation of the *Financial-EDI-Over-The-Internet* pilot project, the Internet related lessons from the *Financial-EDI-over-the-Internet* pilot project do not revolve around the cost and service level concerns of using the Internet versus other transport mechanisms. Rather, it is clear that special attention had to be paid on the organization of Internet based services. Like most large

corporations, BofA is organized into functions such as marketing, product development, customer service, information systems support, and information systems security. When designing and implementing Internet based electronic commerce services, it is no longer clear which function should be responsible for the project. For example, is the firewall ■ part of information systems support, the business unit providing the EDI based service, or should it be ■ responsibility of telecommunications? Moreover, is a corporate WWW site a part of marketing, each business unit, or does it belong to the network support? In the *Financial-EDI-over-the-Internet* project, the organizational issues were solved by forming a project organization which included marketing, the responsible business unit called Global Payment Services, information systems support, information systems security, telecommunications, and the customer partner, Lawrence Livermore National Laboratory (LLNL).

Also during the pilot project, a new unit called Interactive Banking was formed to answer to the customers' increasing requests for Internet based services. The Interactive Banking unit, a company within a company, was founded to overcome some limitations of the functional organization. Instead of attempting to coordinate the corporate functions for electronic commerce, the Interactive Banking unit consists of its own technology, marketing, product development and customer service. Such serviced based concerns are indicative of the infrastructure changes the Internet is bringing to those engaged in the electronic market place.

### 7.4 Open Issues

After the questions concerning security, reliability and speed have been adequately addressed by the *Financial-EDI-over-the-Internet* pilot project, several issues remain open. The open management questions include: what are the arrangements with the EDI business partners concerning electronic signatures, certification, or encryption key management?; what is the policy for offering public WWW services of which EDI service may be a part of?; and can all functions decide about their own Internet policy and implement their own WWW sites? EDI and Internet based services require a new kind of planning to address these questions. This planning is essential, because of immediate and global visibility called "web presence."

BofA has solved the problem by centralizing the Internet management. All units may develop WWW sites for internal use, but only one unit is responsible for the external BofA WWW image and standards.

Other open issues are directly related to the pilot project. First, the data we have is somewhat biased since the organizations only tracked the messages within their systems and not through their transit over the Internet. We do not have much information about the intermediate nodes through which the message packets traveled. Some of this information could be used to explain some of the unresolved problems. These involved message truncation and problems with LLNL's difficulty with decrypting two 824 documents.

Second, as BofA ramps up to handle a greater number of clients, two issues become important beyond building a full-scale hardware-based system:

- The pilot system relied on e-mail messages and intense human attention to track the transit of the messages through the system. This worked during the pilot since transmission occurred only once, maybe twice a day. This system will probably not be feasible once BofA has numerous clients and has to manage many FEDI transmissions a day. The standards for monitoring message sequencing and transmission problems for Internet FEDI are not very well developed (IETF-EDI Working Group, 1993). There is a need and opportunity for an organization to make these procedures explicit and independent of human monitoring as much as possible.
- There is also a need to develop a system for managing the public keys of BofA's clients – to certify them, store them, and apply them to incoming encrypted messages. One option would be to use a package like Premenos' Templar, although this would work only for clients that use Premenos Templar as well. For non-Templar clients, BofA has to develop the procedures and storage facilities for managing these keys, including determining who would serve as a Certifying Authority.

Some critics of EDI over the Net also claim that the projected savings are more apparent than real. Transport costs, a big savings from using the Net, constitute only 5% to 15% of the total cost of a VAN operation, says Brian Dearing, president and CEO of

ARI Network Services Inc., a provider of vertical turnkey services in Milwaukee and a 10-year veteran of the EDI industry. The rest, he says, goes to support and maintenance. That cost will shift to users that start using the Net for EDI. "The deeper we go into the Internet, the more demand there's going to be for in-house management," says Roger Cadaret, chairman of the board of directors at the Data Interchange Standards Association, a not-for-profit corporation in Alexandria, Va., that is the

secretariat for the Accredited Standards Committee X.12 and the administrator for the Pan-American EDIFACT Board. For example, a corporation handling 125,000 messages a month pays \$50,000 to \$100,000, depending on the VAN and how much of the traffic is transmitted during prime time, according to David Raye, VP of operations at the PC division of TSI International in Bannockburn, Ill., which sells PC-based EDI software.

## 8. Conclusion

Because of the pilot project, BofA has gained significant experience in the immediate organizational issues involved in electronic commerce initiatives. For BofA, from a technical standpoint, the *Financial-EDI-over-the-Internet* pilot is a systems integration project. Systems integration today seldom involves novel or unsolvable problems. Thus, the *Financial-EDI-over-the-Internet* project may have significant technical obstacles, yet it is mainly about management and policies.

Retrospectively, the pilot project had a characteristic, which can be called real time research (Porra, 1996). Real time research means that instead of conducting an isolated and insulated laboratory pilot study, the *Financial-EDI-over-the-Internet* pilot project included real organizational units which worked together under real circumstances using real business transactions. Moreover, up to certain dollar amounts, the pilot system was the only system used for FEDI transactions between BofA and LLNL. BofA sees such real time research as a basic part of their experimental method to employ new technologies. Real time research uses the existing organizational resources, people, software, and hardware to manufacture a technological and organizational solution. Compared with more traditional research and development projects, real time research is effective. It solves *real* problems in

actual circumstances by the people who will be responsible for the electronic commerce service in the future. In real time research, the functional organizational unit creates an environment for entrepreneurial teams. They behave like start up businesses within businesses. Reality based experiments like BofA and LLNL demonstrate the feasibility to utilize the Internet as an entrepreneurial rather than laboratory environment.

By choosing to be proactive by stepping into the real-time world of the Internet, BofA has demonstrated ■ strategic shift in its marketing, services, and information systems development methods. As the Financial-EDI-over-the-Internet pilot demonstrates, the openness of the electronic markets will fundamentally change the way companies organize and interact. Preparing for the Internet explosion, both organizationally and technically, BofA is not overly concerned with competition. Banking is a complex business which has both regulated and unregulated areas. Some of the unregulated areas are attractive for companies, such as financial software manufacturers who know the technology and aspects of electronic banking. BofA, however, knows both the regulated and unregulated banking and is able to provide integrated customer services. As such, BofA can guarantee its continued presence in the financial market place by the implementation of the Internet project.

With the validation provided by this *Financial-EDI-over-the-Internet* project, BofA is in position to make strategic changes to its marketing and operational components by incorporating the new Internet solutions in its product/service mix. The strategic lessons are not technical -- they are organizational. Any large firm wishing to examine the Internet for possibilities can be assured that the technical solutions exist. However, without the willingness to make significant organizational, strategic marketing and technical infrastructure changes, the technology of the Internet will have little utility.

## References:

Adhikari, R. (1996). Electronic Commerce – EDI Heads For the Net – Companies Get Faster Messages, Reach New Markets, And Cut Costs. *Information Week*, (May 6).

Bhimani, A. (1996). Securing the Commercial Internet. *Communications of the ACM*, 39, 6 (Jun 1996), 29-31+.

Cohen, F. (1996). Your firewall won't save you. *Forbes*, ASAP Supplement, (Jun 3), 84.

Davis, J., & Parsons, M. (1995). EDI vendors adjust strategies in face of growing Internet. *InfoWorld*, 17/18, 52/1 (Dec 25, 1995/Jan 1, 1996), 39.

Fleishman, G. (1996). *Transactions over the Net*. [Http://maigret.popco.com/hyper/inet-marketing/archives/9501/0120.html](http://maigret.popco.com/hyper/inet-marketing/archives/9501/0120.html). Internet Marketing Mailing List.

Gaffin, A. (1994). *NCD will provide secure banking over the Internet*. [Http://haas.berkeley.edu/~helmer/e...t/artikel/epayment/palyers/fv.html](http://haas.berkeley.edu/~helmer/e...t/artikel/epayment/palyers/fv.html), Network World, Inc.

Green, C. (1993). *EDI: Adoption Rationale and its Relationship to Implementation*. A White Paper. Information Systems Research Center, College of Business Administration, University of Houston.

Hutt, A.E., Bosworth, S., & Hoyt, D. B. (1995). *Computer Security Handbook*. (Third, ed.). New York, NY: John Wiley & Sons, Inc.

Iacovou, C. L., Benbasat, I., & Dexter, A. S. (1995). Electronic data interchange and small organizations: Adoption and impact of technology. *MIS Quarterly*, 19, 4 (Dec 1995), 465-485.

IETF-EDI Working Group. (1993). *Requirements for Interoperable Internet EDI*. Internet Engineering Task Force for EDI (<ftp://sterling.com/edi/lists/ietf-edint/IETF03.txt>).

Lawrence Livermore National Laboratory. (1996). *Financial Electronic Data Interchange Pilot Project*. Lawrence Livermore National Laboratory, Finance Department, UCRL-AR-124103, (May 1, 1996).

Mann, S. (1996). Good-bye EDI, hello Internet. *Manufacturing Systems, Supply-Chain Strategies: Forecasting and Demand Management Supplement*, (Jun 1996), 16A-17A.

Messmer, E. (1995). EDI heavies push data over the "Net". *Network World*, 12 48 (Nov 27, 1995), 1,8.

Messmer, E. (1996). The Internet rocks EDI boat. *Network World*, 13, 18 (Apr 29, 1996), 57.

Ogilve, C. W III (1996). Cyberbanking. *Bank Management*, 72, 3 (May/Jun, 1996), 14-18.

Pant, S., & Hsu C. (1996). Business on the Web: Strategies and Economics. *Computer Networks & ISDN Systems*, 28, 7-11 (May 1996), 1481-1492.

Premenos (1996). *Templar Software and Services – Secure, Reliable, Standards-Based EDI Over The Internet*. [Http://www.templar.net/](http://www.templar.net/), Premenos.

Porra, J. (1996) *Colonial Systems, Information Colonies and Punctuated Prototyping*. Jyvaskyla Studies in Computer Science, Economics and Statistics, 33. Jyvaskyla University Press, Jyvaskyla, Finland.

Segev, A., Wan, D., Beam, C., Toma, B., & Weinrot, D. (1995). *Internet-Based Financial EDI: A Case Study*. The Fisher Center for Information Technology and Management, Institute of Management, Innovation and Organization, University of California, Berkeley, Working Paper CITM-95-WP-1006, (August 1995).

Smith Bers, J. (1996). Diving into Internet EDI. *Bank Systems & Technology*, 33, 3 (Mar 1996), 38-40.

Sokol, P. K. (1995). *From EDI to Electronic Commerce – A Business Initiative*. New York, NY: McGraw-Hill, Inc.

Sterne, J. (1995). *World Wide Web Marketing – Integrating the Internet into Your Marketing Strategy*. New York, NY: John Wiley & Sons, Inc.

Stipe, S. E. (1996). CEO's express skepticism about selling on the Internet. *Best's Review (Life/Health)*, 96, 12 (Apr, 1996), 24-29.

Stuart, A., & Dahle, C. (1996). CIOs at the crossroads. *CIO*, 9, 16 (Jun 1, 1996), 28-30.

Vacca, J. (1996). *Internet Security Secrets*. Foster City, CA: IDG Books Worldwide, Inc.

Wan, D., Beam, C., & Weinrot, D. (1995). *Interview with Bill Jetter and Diana Bowler*, Apr.5, 1995. Haas School of Business, University of California, Berkeley.

Wheatman, V. (1995). *Financial EDI: Getting from Here to There*. Northern California EDI User's Group.

Williamson, M. (1996). Banks on balance. *CIO*, 9, 17 (Jun 15, 1996), 70-80.

## Glossary

820 -- Payment Order/Remittance Advice Transaction Set (820). The 820 transaction set can be used to make a payment send ■ remittance advice or make a payment and send ■ remittance advice. This transaction set can be an order to a financial institution to make a payment to a payee. It can also be a remittance advice identifying the detail needed to perform cash application to the payee's accounts receivable system. The remittance advice can go directly from payer to payee through a financial institution or through a third party agent.

824 -- Application Advice Transaction Set (824). The 824 transaction set can be used to provide the ability to report the results of an application system's data content edits of transaction sets. It is designed to accommodate the business need of reporting the acceptance/rejection or acceptance with change of any transaction set.

827 -- Financial Return Notice Transaction Set (827). The 827 transaction set can be used to report to the originator the inability of the originating financial institution to have the Payment Order/Remittance Advice Transaction Set (820) processed.

997 – Functional Acknowledgment Transaction Set (997). The 997 transaction set can be used to define the control structures for a set of acknowledgments to indicate the results of the syntactical analysis of the electronically encoded documents. This standard does not cover the semantic meaning of the information encoded in the transaction sets.

MIME -- Multi-purpose Internet Mail Extensions. MIME is an Internet Standard that defines the format of email message bodies to allow multi-part textual and non-textual message bodies to be represented and exchanged without loss of information, including video, audio, graphics, and other application specific data.

**MOSS** – MOSS is ■ Privacy Enhanced Mail (PEM) derivative that is ■ Proposed Internet Standard for adding security services to (MIME). It uses the cryptographic techniques of digital signature and encryption to provide origin authentication, integrity, and confidentiality to MIME objects. Users of MOSS can know who originated a message, that the message has not been changed enroute, and that the message was kept secret from everyone except the intended recipients.

**PEM** -- Previously, the Internet community endorsed ■ secure email technology standard called Privacy Enhanced Mail (PEM), which provides authentication, integrity, confidentiality, and non-repudiation of text-based email messages.

# Scalable Document Fingerprinting (Extended Abstract)

Nevin Heintze  
*Bell Laboratories*  
Murray Hill, NJ 07974  
nch@research.bell-labs.com

## Abstract

*As more information becomes available electronically, document search based on textual similarity is becoming increasingly important, not only for locating documents online, but also for addressing internet variants of old problems such as plagiarism and copyright violation.*

*This paper presents an online system that provides reliable search results using modest resources and scales up to data sets of the order of a million documents. Our system provides a practical compromise between storage requirements, immunity to noise introduced by document conversion and security needs for plagiarism applications. We present both quantitative analysis and empirical results to argue that our design is feasible and effective. A web-based prototype system is accessible via the URL <http://www.cs.cmu.edu/afs/cs/user/nch/www/koala.html>.*

## 1 Introduction

As more information becomes available on the internet, searching for documents based on textual similarity is becoming increasingly useful. For example, suppose that I have an early version of a research article, and that I want to determine where it was eventually published (so that I can cite it appropriately), or find if there is a more up-to-date version that fixes previous errors, or perhaps locate an old technical report version that contains more complete

---

The design and implementation of this system was carried out in the Summer of 1995 while the author was at Carnegie Mellon University, supported by the US Postal Service. This work is the opinion of the author and does not necessarily represent the view of his employer or the US Postal Service.

proofs or implementation details. Given that the various versions of an article are likely to have a significant amount of the text in common, a textually related document search is very likely to locate the earlier/later versions of the article (assuming they are accessible to the search engine).

Another application of this kind of search is detection of copyright violations and plagiarism on the internet. It is now all too easy to obtain a paper over the internet, modify the cover page to insert your name in place of the original authors, perhaps change the title and abstract, and submit the paper as if it were your own. Given the large number of conferences and journals, and the imperfections of the refereeing process, the chances of such a paper slipping through undetected are significant. This kind of plagiarism has been successfully carried out in the past and was the subject of a recent editorial in the Communications of the ACM [4]. Arguably it was inevitable that the individual involved would be discovered. What is surprising is the scope of his activities, the time it took before he was discovered, and that he was able to continue with some success even after his case was well known.

More generally, the internet raises major plagiarism and copyright problems because of the ease by which documents may be copied and modified. Resources such as newsfeed wire services, newspaper articles, netnews articles, online books and so forth are increasingly at risk. As a result, many important sources of information are not made available online because the individuals and organizations that own the information find these risks unacceptable. On another front, there is increasing concern about plagiarism of coursework papers at the college undergraduate and graduate level.

There are two approaches to this plagi-

rism/copyright problem. In the first, digital signatures or watermarks are included in a document [3, 10]. These signatures may involve the use of particular word spacings or checksums of components of a document. Unfortunately, these signatures can often be deleted (particularly if the document is translated from one format to another). Moreover, these approaches are not well suited for detecting partial matches involving modified documents. The second approach involves the registration and storage of documents and subsequent textual matching with new documents to track copies and modified versions [1, 6, 11]. This approach has been used in a number of implemented systems [2, 7, 8, 9].

In this paper we consider the general problem of textual matching for document search and plagiarism/copyright applications. We focus on the question of how to build a practical online system that provides reliable search results for a data set of the order of a million documents, while using modest resources (0.5G of disk, i.e. about 500 bytes per document). Three central issues arise: first, how can we meet the space constraints; second, how can we provide reliable search in the presence of noise (to perform textual matching, we must convert documents into text, and this is typically a very noisy and unreliable process), and third, how can we meet the security needs of plagiarism applications (if it is easy to circumvent the matching of related documents by making a handful of selective text changes, then the system will be of little value for plagiarism detection). We present both quantitative and empirical analysis of our system. A web-based prototype system called Koala is accessible via the URL <http://www.cs.cmu.edu/afs/cs/user/nch/www/koala.html>.

## Related Work

Most closely related to our work is the Stanford Digital Library work on SCAM [7, 8, 9]. Their approach uses relative word frequencies. Similarity between documents is based on a modified cosine similarity measure. The storage requirements for this approach are approximately 30%-65% of the size of the original documents ([9] reports data storage requirements of between 37MB and 79MB for a document set of 120MB, depending on the chunking approach used). The use of words as the basis for document analysis requires textual representations of documents that faithfully preserve word boundaries.

Our approach is based on selecting a set of subse-

quences of characters from a document and generating a fingerprint based on the hash values of these subsequences. Similarity between two documents is measured by counting the number of common subsequences in fingerprints (the reliability of this measure is critically dependent on how the subsequences are selected from a document). One major difference from SCAM is that we store less than 500 bytes per document (400 bytes for the actual fingerprint, about 20 bytes for document identification such as URL and email information, and some overhead for indexing), which typically means that our storage requirements are about 0.5-1% of the size of the original documents, almost two orders of magnitude less than the requirements for SCAM. Another significant difference is that we accept documents in a variety of different formats (including Postscript generated from TeX, PageMaker, Microsoft Word and FrameMaker). To perform textual comparison on such documents, we must first convert them to text. The problem is that such conversions introduce many errors. In particular, punctuation and word spacing are very unreliable. An early version of our work was word based, but did not give satisfactory results. Instead, we have adopted techniques that are largely insensitive to word boundaries and other common errors introduced by document conversion.

The idea of selecting a set of pieces of a document and then hashing these to obtain a document fingerprint is used by Manber in *sif*, a tool for finding similar files in a large file system. However, the motivations for *sif* and our work are very different. First, *sif* focuses on similarities of 25% and higher, whereas we strive to provide reliable information for matches of 3-5% and lower (the key is to reduce the number of false positives while retaining important matches). Second, our work addresses the problem of tolerance to noise. Third, our work addresses special issues related to plagiarism applications. As a result, the *sif* and Koala system designs differ quite substantially.

## 2 Textual Relationships

We first address the question of what kinds of textual relationships should be considered significant. Checking for exact matches is easy, but not satisfactory. For example it would miss matches where one document is the result of minor edits of the other. It would also miss identical documents that differ because of noise introduced by document translation processes

(Postscript to text conversion, OCR, etc.). Moreover, it would ignore most of the interesting textual relationships between documents. In this paper, we consider the following general kinds of relationships to be significant:

1. Identical documents.
2. Documents that are the result of small edits/corrections to other documents.
3. Documents that are reorganizations of other documents.
4. Documents that are revisions of other documents.
5. Documents that are condensed/expanded versions of other documents (e.g. journal versus conference versions of papers).
6. Documents that include portions (say several hundred words) of other documents.

We require that the first five classes of relationships be identified with very high probability; for the remaining class we will tolerate a small number of false positives and false negatives.

## 3 Fingerprinting

### 3.1 Full Fingerprinting

Consider the following simple fingerprinting scheme: given a document, let the fingerprint of the document consist of the set of all possible document substrings of length  $\alpha$ . There are  $l - \alpha + 1$  such substrings, where  $l$  is the length of the document. Comparing two documents under this scheme is simply a matter of counting the number of substrings common to both fingerprints: if we compare a document  $A$  of size  $|A|$  against a document  $B$ , and if  $n$  is the number of substrings common to both documents then  $n/|A|$  is the measure of how much of  $A$  is contained in  $B$ . If  $\alpha$  is chosen appropriately, this simple fingerprint gives reliable document matching results. We refer to this scheme as *full fingerprinting*. Although it is not practical (for space reasons), it is a very useful measure of document similarity, and we shall use it for the evaluation of our system in Section 7. (We remark that it would not be a good idea to construct fingerprints by chopping up the document into  $\text{floor}(l/\alpha)$  substrings

by making a cut at every  $\alpha^{th}$  character, because insertion of a character at the start of the document would shift the substrings by 1 and the resulting fingerprint would be a poor match to the original, even though the two documents are almost identical).

The choice of  $\alpha$  in this fingerprinting scheme is particularly important, and is subject to two conflicting constraints. If  $\alpha$  is too small, then there will be many false matches (e.g. if  $\alpha$  is the size of a word, then the scheme reduces to little more than comparing lists of words in documents, a poor similarity metric). If  $\alpha$  is too large, then there will be many false negatives because one character change can affect  $\alpha$  substrings in the fingerprint (e.g. if  $\alpha$  is the size of a paragraph, then a single character change in a paragraph would effectively prevent matching for the entire paragraph). We remark that there is no “right” value for  $\alpha$ : we cannot quantitatively or empirically calculate a definitive value for  $\alpha$ . In essence, the choice of  $\alpha$  defines the notion of document similarity for the system. Different values of  $\alpha$  will be useful for different kinds of searches. The value of  $\alpha$  used in this paper is effectively around 30–45 (more precisely, our strings consist of 20 character sequences of consonants; we discuss this in detail in Section 5). We investigate the effect of different values of  $\alpha$  in Section 7.

### 3.2 Selective Fingerprinting

As mentioned earlier, full fingerprinting is conceptually useful but it is not practical because of the sizes of the fingerprints generated. To reduce the size of a fingerprint, we select a subset of the substrings from the full fingerprint. Since the goal of our work is to treat documents that vary in size from several thousand words to several hundred thousand words while meeting very tight space constraints, we have chosen to select a fixed number of substrings, independent of the size of the document. We call this *fixed size selective fingerprinting*. (An alternative is to select a fixed proportion of the substrings, so that the size of the selective fingerprint is proportional to the size of the document. The main drawback of this alternative is space consumption: to provide accurate fingerprinting of documents with several thousand words we would need a fingerprint containing 50–100 substrings, and this means fingerprints of size 5000–10000 for documents containing several hundred thousand words.)

The design a fixed size selective fingerprinting sys-

tem revolves around two choices: fingerprint size and selection strategy (that is, which substrings do we select from the full fingerprint). We discuss these in the next two subsections.

### 3.3 Fingerprint Size and Security

We employ different size fingerprints for storage and search: the fingerprints we store in the database have size 100, but the fingerprints used for searching have size 1000. Importantly, the search fingerprint for a document is a strict superset of the fingerprint used for storage. There are two reasons for this choice. The first is reliability, and is intimately connected with design decisions discussed in Subsection 6.1. The second motivation is security: we want our system to be resilient under attack by would-be plagiarists. To illustrate the issue, first suppose that we use fixed size fingerprints of 100 for both storage and search and that the selection strategy is publicly known. In this case, it would be easy for a plagiarist to determine which 100 substrings are part of the fingerprint, and make 100 changes at the appropriate places in the plagiarized version so that it no longer matches the original. If, instead of making the selection strategy public, we keep it secret (for example, we could use some secret seed value to guide the selecting strategy), then by a trial and error process, it is still possible to find an appropriate set of 100 changes (for example, one could chop the original document into pieces and search separately on these pieces to identify the selected substrings).

We provide better security by periodically changing the stored fingerprint of a document. The use of two fingerprints provides a particularly convenient way to achieve this: we obtain a new stored fingerprint by simply choosing a different subset of the search fingerprint (since the ratio of sizes involved is 100:1000, this still gives considerable scope for change). The advantage of this approach is that we do not need to change the search engine (i.e. we still generate the same search fingerprint) to search against the modified stored fingerprint. This is important, because it allows us to change the database incrementally: we can update the stored fingerprints of a few documents at a time in a transparent manner. To support this process, we maintain a list of URLs for each fingerprinted document so that we can retrieve the document and recompute its fingerprint as desired. We also maintain a contact email address for each document to help resolve stale URLs. We envi-

sion updating fingerprints on a regular basis (perhaps once every six months or year), with irregular updates if there is suspicious activity relating to a document (such as an unusually large number of searches for it).

### 3.4 Selection Strategy

One simple strategy is random selection. However this gives poor results. For example suppose that we have a document of length 50,000 (which gives rise to about 50,000 possible substrings of length  $\alpha$ ) and we use 100 substring fingerprints for storage and 1000 substring fingerprints for search. Now consider matching the document against itself. The probability that any particular substring appears in the storage fingerprint is  $100/50000 = 1/500$ . Hence, the expected number of substrings from the search fingerprint that match the storage fingerprint is  $1/500 \times 1000 = 2$  (i.e. a match ratio of about 2%). The results are of course much worse for documents that are related but not identical.

To provide more reliable matches, the selection strategy must select similar substrings from similar documents. One approach is employ a string hash function, and then a fingerprint of size  $n$  can be obtained by picking the  $n$  substrings with the lowest hash values. The approach we use is related to the hash function approach and gives similar results, but reduces false positives. We defer the details to Section 6.

### 3.5 Limitations of Fixed Length Fingerprints

An important measure of the reliability of a fingerprinting scheme is how closely its results correlate with those from full fingerprinting. For the fixed length selective fingerprinting approach we have chosen, the correlation is good for documents of similar size, but can become problematic for documents of significantly different size. Specifically, we can show that for documents of identical size, the expected match ratios for fixed length selective fingerprinting are identical to those for full fingerprinting. However for documents of different size, the results can vary by a ratio as high as the ratio of sizes of the two documents. To illustrate the problem, consider the extreme case of matching a document of 1000 words against a document of 100,000 words, and suppose that the smaller document appears once in the larger

document. Now if the stored fingerprint of the larger document has size 100, then on average there will be one substring for every 1000 word piece of the larger document. In other words, the stored fingerprint of the larger document will have about one substring in common with the smaller document (i.e. about a 1% match ratio), compared with the 100% match given by full fingerprinting. We are currently investigating ways to address this issue, including using variable sized fingerprints and flagging low match ratios as significant if document sizes vary significantly.

## 4 Fingerprint Storage

Fingerprints are hashed and stored using a very simple indexed file. Specifically, each substring selected for inclusion in a fingerprint is hashed to a 28-bit unsigned integer. The top 16 bits of this 28-bit hash values are used as an index into a table at the start of the file. This table consists of pointers into 256 word blocks. The first word of each block contains a chain pointer to an overflow block (if there is one), and the second word contains a count of the number of words used in the block. The remaining 254 words are used to store fingerprint entries: each entry consists of the lower 12-bits of the 28-bit substring hash value and a 20-bit document identifier (for a maximum of 1M documents), which gives a total of 32 bits, or one word per fingerprint substring. Since we have 100 substrings per (stored) fingerprint, each fingerprint occupies 400 bytes.

We also store a log of each document in the database. This log includes the document identifier, date, URL and a contact email address. Currently this information is stored as raw ascii and consists of about 50-80 bytes per document. This can be compressed substantially. Early experiments indicate that a factor of 4 should be possible.

The simple indexing scheme we have used has a number of drawbacks. First, if the file only contains a small number of fingerprints then there will be a lot of wasted space because most of the blocks will be nearly empty. Second, as more documents are added, the overhead of following overflow blocks can become significant. For example, if there are 1M documents, there will be 100M words of fingerprints, which will occupy about 400-500K blocks, giving rise to an average chain length of 6-8, or about 6000-8000 disk probes per document search (search fingerprints have size 1000). These issues can be addressed by more

sophisticated indexing/disk-management schemes.

## 5 Document Noise

To generate fingerprints and perform document matching, we must first obtain text versions of documents. Unfortunately, this is an unreliable process that introduces many errors. One of the main formats we wish to support is Postscript. Postscript interpreters can be adapted to produce text, but they are typically slow and often produce poor results. Alternatively, for Postscript output by specific tools, we can exploit the format of Postscript generated by the tool to recover the text quickly and fairly accurately (this is the case for example with TeX/dvips generated Postscript). The problem is that the formats change as the tools evolve, and we need different programs to deal with different Postscript tools.

For Postscript conversion, the main errors introduced involve punctuation, non-alphabetic characters and spacing. In particular, word boundaries are often distorted. There are some secondary problems with vowels and uppercase/lowercase distinctions. We factor out these problems by ignoring all but non-vowel characters and converting everything to lower case. This allows us to use fast Postscript to text converters based on string extraction (the translator we use is a modified version of Jason Black's ps2txt program, which in turn is based on a program by Iqbal Qazi). By focusing on non-vowel characters and converting to lower case, we have obtained very reliable results for Postscript generated from TeX, PageMaker, Microsoft Word and FrameMaker. Note that by considering only consonants, our approach is not actually based on document substrings, but rather on character subsequences of the original document. We use subsequences of length 20, and given the typical distribution of consonants, this corresponds to spans of about 30-45 characters in the original document.

## 6 Reducing False Positives

One of the goals of our system is to provide reliable low level match information, and in particular, to reduce the number of false positives. This is important for a number of reasons. First, the identification of low level matches appears to be an interesting search paradigm for locating related documents. Second, it helps offset some of the limitations of fixed length fingerprints. Third, it has important performance im-

plications: in the context of a database of millions of documents, false positives can significantly increase the cost of searching.

## 6.1 Fingerprint Generation

The selection strategy used to choose the substrings to include in a fingerprint can have a significant impact on the number of false positives. The issue is that some substrings occur significantly more frequently than others – sometimes by many orders of magnitude. Moreover, these frequent substrings are more likely to be selected in a fingerprint if fingerprint construction does not consider substring frequency. To illustrate the potential impact of this, first suppose that each substring is equally likely to appear in a document, and consider comparing two unrelated documents. If the hash function used is well behaved, then the search fingerprint contains 1000 randomly selected elements from a space of  $2^{28}$  elements and the storage fingerprint contain 100 randomly selected elements from the same space. Now, the probability that at least one element from the  $2^{28}$  space will appear in both fingerprints is given by the probability that a particular element appears multiplied by the number of possible elements. This can be approximated by:

$$\left( \frac{100}{2^{28}} \times \frac{1000}{2^{28}} \right) \times 2^{28}$$

which is about 0.00037. Hence, for a database of  $1M$  documents, we can expect about 370 random noise hits for each search.

Now suppose that a fingerprint consists of substrings that are 10 times more frequent than the average. Then the probability of a one substring match between two documents increases by a factor of  $10 \times 10$  to 0.037, or near 40K random noise hits for a search against  $1M$  documents.

We can significantly reduce false positives by using a substring selection strategy based on frequency measures. One way to do this is to compute the set of all substrings for the document and then pick the least frequently occurring substrings. However this is computationally expensive and does not yield good results because the space of substrings in one document is not a useful indication of the overall frequency of substrings.

Instead, we use a frequency measure based on the first five letters of a substring. This is not only cheaper to compute, but gives useful results. The intuition is that the distribution of five letter se-

quences in a specific document is a useful approximation to the general distribution of five letter sequences. Hence if we pick substrings whose first five letters occur infrequently in a document, it is likely that the first five letters of these substrings will occur infrequently in general. Substrings whose first five letters occur infrequently are likely to be such that the entire substring occurs relatively infrequently. In Section 7 we give experimental evidence that indicates this technique can reduce false positives by more than a factor of two.

Underlying this approach is the assumption that the substring frequency distribution of (significant) overlapping text segments does not vary substantially from the frequency distribution of other text segments (which would imply that match ratios for related documents are not affected by focusing on infrequent substrings). This assumption appears to be valid in practice.

One problem with the use of five letter frequency distributions for substring selection is that different documents may have slightly different five letter frequency distributions. The use of different size fingerprints for storage and search provides an effective way to address this. To illustrate why, consider a substring  $s$  that is common to two documents  $A$  and  $B$ . If  $s$  is selected in  $B$ 's stored fingerprint (that is,  $s$  is “very infrequent” according to  $B$ 's frequency measure), then although it may not appear in  $A$ 's stored fingerprint (because the frequency measures for  $A$  and  $B$  differ), it most probably will appear in  $A$ 's (much larger) search fingerprint because it is likely to be “moderately infrequently” according to  $A$ 's frequency measure.

We remark that another way to identify infrequent substrings is to use the fingerprint database to provide frequency measures. One disadvantage of this approach is that fingerprint generation can no longer be performed as a stand-alone operation. However it has the potential to very reliably identify infrequent substrings and deserves further investigation.

## 6.2 Fingerprint Matching

Some substrings are very common in certain collections of documents. For example, in a technical report series, substrings generated from addresses, funding agencies acknowledgements and strings such as “This work is the opinion of the author and does not necessarily represent the view of his employer or ...” appear in many documents. Such substrings are

difficult to recognize within the context of a single document (and so the technique described in the previous section does not detect them), but require the context of a collection of documents.

We use a frequency check during search to isolate these strings. When a search is performed and a particular (hashed) substring is looked up in the database, we check to see if this particular string appears in 4 or more documents. If it does, then we ignore it during the search. However, this raises a security issue: if one could repeatedly add copies of the one document to the database, then eventually all of the substrings of the document would be ignored, and the document would not generate a match. To address this situation, we cap the number of ignored substrings at 10. In Section 7, we show that this technique can reduce false negatives by between 15% and 85%, depending on which of the other checks in this section are deployed. We remark that the values of 4 and 10 deserve further evaluation using different sizes and classes of document sets. We also remark that this technique is an application of a very standard information processing idea: discount the significance of common features.

### 6.3 Document Prologs

While the use of frequency checks provides one way to ignore common substrings, other techniques can also be useful. In particular, most of the problematic strings such as addresses, funding agencies acknowledgements etc., appear at the start of a document. Also, when a Postscript file is converted to text, the first words of text are often from the preamble of the Postscript file and indicate the tools used to generate the file; they have nothing to do with the actual text of the document.

One simple approach is to ignore the first part of a document. In Section 7 we show that ignoring the first 1000 characters of a document gives useful reductions of false negatives without significantly affecting other matches. Moreover, it is useful in tandem with the technique described in the previous subsection.

## 7 Results

We now present some experimental results from an implementation of the system. The main data set we use is a collection of 366 technical reports from the Carnegie Mellon University School of Computer Science (our set essentially consists of all reports avail-

able online as of August 1995). This set consists of just over 30MB of text. Table 1 gives the distribution of matches when each document is searched against all others. The number in parenthesis in the right hand column is the number of non-identical document matches (i.e. 372 - 366). Note that there were 763 matches at the 1% level, which is about two 1% matches for each document. This is higher than expected. It reflects the fact that the data set has a high degree of low-level correlation: it is generated by a relatively small group of people with shared experiences and background (for example, people in this group tend to cite each other's work). Some of these low-level commonalities would be removed by the technique described in Section 6.2, however the data set is too small for this technique to remove all of them. The web-based implementation of the system has a larger database (about 3000 documents) that includes the technical report data set. When a technical report is matched against this larger database, we typically find twice as many 1% matches as we obtain when matching against just the technical report database. This means that the 2600 other documents are generating about as many 1% matches as the technical reports (i.e. a 1% match rate that is 7 times lower). This is partly because this data set is larger, but also because it does not have the same degree of low-level correlation (the 2600 "other" documents are technical reports and papers from a wide variety of different institutions).

Table 2 compares fixed size selective fingerprinting with full fingerprinting for a small collection of documents, and provides evidence of the reliability of selective fingerprinting. The left hand column gives the match ratios reported by our system, and the right hand column gives the results for full fingerprinting as both a match ratio and as raw data (common-substrings/total-substrings). For this small collection of documents, selective fingerprinting gives match ratios that are within a factor of two of full fingerprinting, and usually much closer.

To establish the utility of the techniques for reducing false positives, we present two collections of data. Table 3 considers both (a) dropping frequent substrings during searching and (b) cutting the first 1000 characters of a document. The first line of the table gives the match ratio distribution for matching all documents against all documents with both techniques (a) and (b) enabled. The second line disables just (a), and the third line disables just (b). The final line disables (a) and (b). Both techniques

match range	1%	2%	3%-5%	6%-10%	11-20%	21-50%	more than 50%
document count	763	98	53	25	15	23	372 (6)

Table 1: Match Distribution for CMU-SCS Technical Report Data Set.

selective fingerprint match ratio (%)	full fingerprint	
	match ratio (%)	matches/total
45	57	13206/22994
9	8.7	2005/22994
5	12	2766/22994
29	55	22541/41010
1	0.01	5/41010
1	0.01	5/41010
1	0.2	89/41010
1	0.08	34/41010
3	2.6	670/25386
1	3.0	782/25386
0	0	0/22994
0	0.3	76/22994
0	0.03	8/22994
0	0.16	38/22994
0	0.19	45/22994

Table 2: Comparison of Selective Fingerprinting and Full Fingerprinting.

are very useful in isolation, but it is clear that they address overlapping issues. They are, however, sufficiently different to be useful in tandem.

Table 4 considers the effectiveness of focusing on infrequent substrings during fingerprint generation (Subsection 6.1). The first line give the baseline match ration distribution, and the second line gives the same results when substrings are selected without regard for frequency. This table indicates a reduction of false positives by more than a factor of two.

The next two tables investigate the effect of fingerprint size. In Table 5, the size of the stored fingerprint is varied from 10 to 500 (the baseline value is 100), while the search fingerprint remains constant at 1000. In Table 6, the size of the search fingerprint is varied from 100 to 5000 (the baseline value is 1000), while the search fingerprint remains constant at 100. The results indicate that our system is surprisingly insensitive to changes in fingerprint sizes. The use of 100 for storage and 1000 for searching yields a good trade-off between search reliability and the level of positive matches.

Finally, table 7 shows the effect of changing  $\alpha$ , the length of character subsequences, from 10 to 50 (the baseline value is 20). As expected, decreasing  $\alpha$  has the effect of significantly increasing the number of low level matches. Increasing  $\alpha$  has the effect of decreasing both the number of low-level and high-level matches.

## 8 Conclusion

We have presented a system for document comparison based on textual similarity. Target applications include related document searches and copyright/plagiarism protection. Our system uses fixed size selective fingerprints based on document substrings, and supports reliable and accurate document comparison with very small fingerprints (about 400 bytes per document). The main novelties of our work are (a) very low storage requirements (almost two orders of magnitude less than competing systems), (b) resilience to noise in documents (such as that introduced by conversion from Postscript to text), (c) security measures to the improve dependability of plagiarism searches in the context of an active adversary, and (d) significant reduction of false positives.

## References

- [1] C. Anderson, "Robocops: Stewart and Feder's mechanized misconduct search", *Nature*, 350(6318):454-455, April 1991.
- [2] S. Brin, J. Davis and H. Garcia-Molina, "Copy Detection Mechanisms for Digital Documents", *Proceedings of the ACM SIGMOD Annual Conference*, May 1995.
- [3] J. Brassil, S. Low, N. Maxemchuk and L. O'Gorman, "Electronic Marking and Identification Techniques to Discourage Document Copying", *Journal on Selected Areas in Communications*, Volume 13, Number 8, October 1995.
- [4] P. J. Denning, "Plagiarism in the web", Editorial, *Communications of the ACM*, 38(12), December 1995.
- [5] U. Manber, "Finding similar files in a large file system", *Proceedings of the 1994 USENIX Conference*, pp. 1-10, January 1994.
- [6] A. Parker and J. O. Hamblen, "Computer algorithms for plagiarism detection", *IEEE Transactions on Education*, 32(2):94-99, May 1989.
- [7] N. Shivakumar and H. Garcia-Molina, "SCAM: A Copy Detection Mechanism for Digital Documents", *Proceedings of the 2nd International Conference on Theory and Practice of Digital Libraries*, 1995.
- [8] N. Shivakumar and H. Garcia-Molina, "The SCAM Approach to Copy Detection in Digital Libraries", *Diglib Magazine*, November 1995.
- [9] N. Shivakumar and H. Garcia-Molina, "Building a Scalable and Accurate Copy Detection Mechanism", *Proceedings of the 3rd International Conference on Theory and Practice of Digital Libraries*, 1996.
- [10] N. R. Wagner, "Fingerprinting," *Proceedings of the 1983 Symposium on Security and Privacy*, pp. 18-22, April 1983.
- [11] D. Wheeler "Computer networks are said to offer new opportunities for plagiarists", *The Chronicle of Higher Education*, pp. 17, 19, June 1993.

	1%	2%	3%-5%	6%-10%	11-20%	21-50%	more than 50%
baseline	763	98	53	25	15	23	372 (6)
freq. check off	896	107	53	25	15	23	372 (6)
include start	2618	194	100	32	15	19	374 (8)
both disabled	17598	2415	334	35	17	19	374 (8)

Table 3: Reducing False Positives I: frequency checks (search) & document preamble.

	1%	2%	3%-5%	6%-10%	11-20%	21-50%	more than 50%
baseline	763	98	53	25	15	23	372 (6)
random	1634	177	104	43	17	29	376 (10)

Table 4: Reducing False Positives II: use of infrequent substrings in fingerprints.

stored fingerprint	1%	2%	3%-5%	6%-10%	11-20%	21-50%	more than 50%
10	—	—	134	—	12	17	370 (4)
20	—	—	217	—	19	16	370 (4)
50	—	442	—	43	39	18	21
100	763	98	53	25	15	23	372 (6)
200	1507	43	49	21	17	19	372 (6)
500	2162	32	38	25	14	18	372 (6)

Table 5: Effects of Varying Stored Fingerprint Size.

search fingerprint	1%	2%	3%-5%	6%-10%	11-20%	21-50%	more than 50%
100	364	54	36	22	20	18	374 (8)
200	520	66	50	19	17	24	371 (5)
500	777	89	52	30	17	20	372 (8)
1000	763	98	53	25	15	23	372 (6)
2000	976	91	63	21	15	20	373 (7)
5000	1064	101	62	32	16	21	372 (6)

Table 6: Effects of Varying Search Fingerprint Size.

$\alpha$	1%	2%	3%-5%	6%-10%	11-20%	21-50%	more than 50%
10	6239	363	105	33	24	22	375 (9)
20	763	98	53	25	15	23	372 (6)
30	411	70	49	22	10	20	371 (5)
40	366	55	42	22	15	17	371 (5)
50	231	37	42	19	13	15	370 (4)

Table 7: Effects of Varying Substring Length.

# A Protocol for Secure Transactions

Douglas H. Steves  
dhs@cs.utexas.edu

Chris Edmondson-Yurkanan  
dragon@cs.utexas.edu

Mohamed Gouda  
gouda@cs.utexas.edu

*Department of Computer Sciences  
The University of Texas at Austin  
Austin, TX 78712*

## Abstract

*Secure transactions form the computational basis for electronic commerce. Many forms of commerce depend upon there being a defined and verifiable relationship between messages in a transaction. We have identified three such relational properties: causality, atomicity and isolation.*

*Causality is a new property. It allows the receiver of a message to deduce and verify the sequence of messages sent and received by the sender prior to that message.*

*In this paper, we present a secure transaction protocol that provides relational properties in addition to the normal properties of secure messages.*

## 1 Introduction

A *transaction* is a partial order of messages between two processes. Transactions proceed in three successive phases:

**initialization** The transaction is accepted by both processes and all parameters are initialized.

**exchange** One or more data messages are exchanged between the two processes.

**termination** The transaction is concluded, either successfully or unsuccessfully.

A message is secure if and only if the following properties can be established as required:

**authentication** The receiver of a message can establish the unique identity of the sender.

**privacy** The message data can only be observed by the sender and receiver.

**integrity** The message data can only be altered by the sender without detection.

A transaction is secure if and only if all messages in the transaction are secure and the following relational properties can be established as required:

**causality** The receiver of a data message can deduce and verify the sequence of previous messages that were sent or received by the sender.

**atomicity** Either all messages in a transaction may be referenced by other transactions, or none may be.

**isolation** The messages in a transaction cannot be referenced by other transactions until and unless the first transaction terminates successfully.

A *secure message* protocol is used to exchange singular messages between two processes with one or more of the secure message properties. A *secure transaction* protocol, on the other hand, is used to exchange related secure messages between two processes with one or more of the secure transaction properties.

The secure message properties defined above are well-known, and are already provided in several existing security protocols, such as SET [11], PGP [8] and SSL [2].

The relational properties of atomicity and isolation are also familiar from classical database theory [4], where they provide for computational separation of transactions. In secure transactions we seek a similar effect, so that transactions that involve exchanges of things of value (cash or commodities) occur atomically (all exchanges occur or none do) and in isolation (no item can be exchanged unless it is received in a successful transaction).

The other relational property, causality, allows each process in a transaction to deduce and verify the order in which data messages were sent and received in the transaction. If some of these data messages constitute a negotiation between the processes of the transaction, then causality ensures that each process can prove exactly how the negotiation progressed.

Next we discuss mechanisms that can be used to ensure these relational properties.

## 1.1 Causality

Ordering is a necessary precondition to ensure causality. Since messages are the only actions in our system, a message can only be caused by a message that preceded it.

Ordering can be established by including an authenticated sequence number or virtual time stamp field in each message. This would, however, not be sufficient, since the sequence number in a message would only acknowledge the reception of a certain number of messages prior to the transmission of the message; it would not, however, demonstrate the reception of *specific* messages, since the inclusion of a sequence number or virtual time stamp does not acknowledge the contents of the previous messages.

What is required is a protocol that provides for proof of message reception. Message reception could be proven by including within each message a copy of all previous messages. While this would certainly be sufficient to prove causality, it would also be inefficient, especially for extended transactions. More importantly, it might also violate the privacy policy for multi-party transactions.

Instead, we include in each message a cryptographically secure message digest of all previous messages sent or received by the sender of the message. This does not violate any privacy concerns, nor is it inefficient since the size of the digest is relatively small and it can be incrementally computed. We define this message digest as *context*.

Unfortunately it is not always possible to provide causality in every transaction. A process  $P$  that sends two messages to another process  $Q$  and then receives a response from  $Q$  cannot validate the context since  $P$  cannot know whether or not  $Q$  received one or both messages prior to sending its response.

Fortunately it is possible to modify the protocol in order to provide causality in situations like this, by employing a half-duplex protocol that allows both processes to track the state of the transaction consistently. While normally this restriction would be a hindrance to performance, transactions requiring negotiation are essentially half-duplex since each process must wait for the other's latest message in order to formulate its response.

## 1.2 Atomicity and Isolation

Atomicity and isolation, are ensured in classical database transactions by using a commit protocol. This protocol is used to turn temporary storage into permanent storage - that is, the updates to the transaction's data are not visible until the commit protocol validates the stored data. Note that it is the pres-

ence of a commit record in the database log which effectively validates the transaction's data.

With secure transaction protocols, of course, we are not concerned with the correctness of data in persistent storage, but rather that data messages are not used or referred to unless their corresponding transaction has been properly committed. This can be accomplished by:

- 1) including a transaction identifier within each data message that effectively ties the message to the transaction
- 2) using a commit protocol to conclude each transaction
- 3) requiring a valid commit record for a transaction prior to the use of a message from that transaction.

The protocol that we define in this paper addresses the first two of these; the last would require the complete definition of the system to ensure that all message references are properly constrained.

Two other types of secure message atomicity protocols are discussed in [10]. *Encryption-based* protocols guarantee atomicity by having processes encrypt all messages that they send, and then using a trusted third party to distribute the keys once all messages are acknowledged by the receiver. *Authority-based* protocols guarantee atomicity by having processes send all transaction messages to the trusted third party, who will then forward them once all messages in the transaction have been received.

Neither of these is suitable, however, for transactions in which earlier messages must be read before a response can be sent. The use of a database-like commit protocol effectively enables this, since it allows write or send operations to depend upon previous read or receive operations.

In the next section we present a protocol that allows two processes to conduct secure transactions. This protocol extends the privacy, integrity and authentication properties of secure messages to also include causality, atomicity and isolation. This protocol is based on trustful relationships, so the properties are not provable to a trusted third party; they can, however, be confirmed by either party, which allows security violations to be detected prior to commitment of the transaction.

In later work, we extend this protocol to model distrustful and multi-party relationships, where the protocol can be used to prove correct transactions and to disprove incorrect transactions, and we provide an implementation of both models.

## 2 Secure Transaction Protocol

### 2.1 Overview

The Secure Transaction Protocol (STP) is a two party protocol that provides authentication, integrity, privacy,<sup>1</sup> atomicity, isolation and causality. In STP, a process may engage in many concurrent transactions with other system processes. Each process in the system has a unique identity (*UID*) and a set of *identity certificates*. The identity certificate used for a transaction is a function of the *UID* of the other process in the transaction.

Logically, an identity certificate for a process contains the *UID* and a *signature verification key* for that process. The identity certificate is also used to derive the *signature key* used by the process itself for the transaction. The sender uses the signature key to sign (authenticate) each message, while the receiver uses the verification key to validate that signature. The concepts of signature key and signature verification key map easily to either asymmetric or symmetric cryptography.

Each transaction in the system has a unique identifier (*TID*). For each transaction in which a process participates, the process stores the *UID* of the other process, the data messages sent and received during that transaction and the state of the transaction. The process also stores the other process's signature verification key to be used during the transaction.

Last, the process maintains the *context* for each active transaction. It is computed as the signed, cryptographic message digest of all data sent or received thus far in the transaction. It is included in all data messages so that the data being sent is effectively represented as the result of previous messages.

During the initialization phase of a normal transaction, one process attempts to begin a transaction with another. The second process may accept or reject the transaction.

If the second process accepts, the transaction enters the exchange phase. The processes begin transmitting data alternately, using a half duplex protocol controlled by a send token.

When the data transfers are complete, the transaction enters the termination phase. If the process with the token wishes to commit the transaction, it sends the other process a commit message. If the second process agrees, it returns its own commit message and the transaction is terminated successfully.

These message transfers are depicted in Figure 1.

<sup>1</sup>Privacy has been omitted from the following specification for readability.

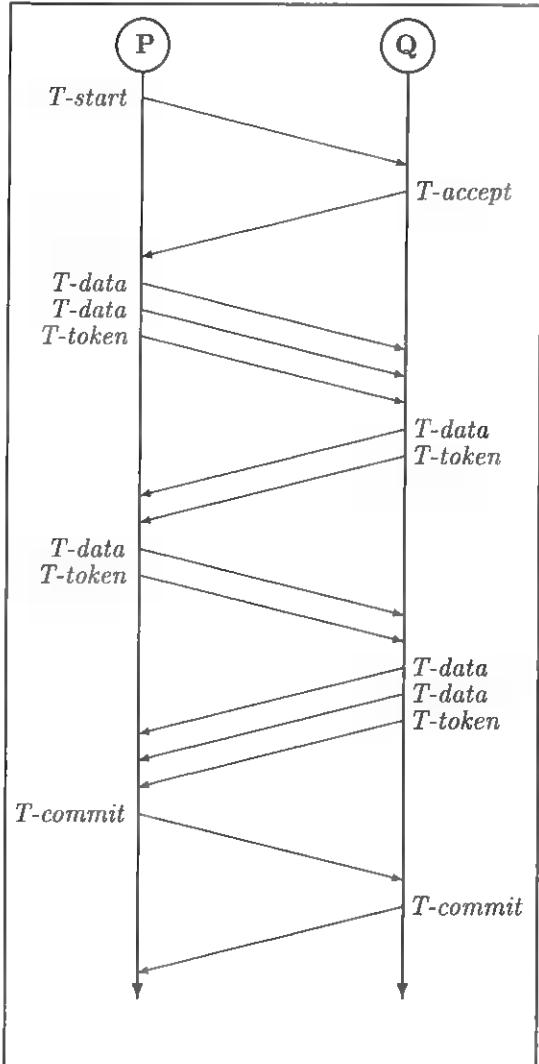


Figure 1: Basic protocol

This protocol assumes that the underlying message transport mechanism will provide reliable, ordered communication except in the presence of persistent media failures or in the event of an adversarial attack. If the communication channel cannot confirm the correct delivery of a message, it will return a fault exception to the process. Messages deleted by an adversarial process will be detected by the channel, while inserted or altered messages must be detected by the receiving process.

In the following sections, we define **STP** in terms of protocol messages, state and transitions. Messages define the state which can be communicated between processes, state defines discrete and specific points in protocol processing, and transitions define the relationship between messages and state. Lastly, we define the protocol itself, using the Abstract Protocol language, from [3].

## 2.2 Protocol Messages

Our protocol contains the following message types:

### **transaction start**

The *T-start* message is used to begin a transaction. This message contains:

- 1) the transaction identifier (TID)
- 2) the identity certificate of the sender
- 3) the signature

### **transaction accept**

The *T-accept* message is sent in response to a *T-start* message if the process decides to accept the transaction. This message contains:

- 1) the TID from the *T-start* message
- 2) the process's own identity certificate
- 3) the signature

### **transaction data**

Once the transaction has been established, the processes may exchange data with *T-data* messages. These messages contain:

- 1) the TID
- 2) the data
- 3) the context
- 4) the signature

### **transaction token**

Communication in **STP** is half duplex. When one process is through sending *T-data* messages, it will send a *T-token* message to allow the other process to transmit. This message contains:

- 1) the TID
- 2) the context
- 3) the signature

### **transaction commit**

When a process wishes to conclude a transaction successfully, it sends a *T-commit* message to the other process. This message contains:

- 1) the TID
- 2) the context
- 3) the signature

### **transaction abort**

When a transaction wishes to terminate a transaction without commitment, it sends a *T-abort* message to the other process. This message contains:

- 1) the TID
- 2) the signature

### **transaction fault**

If there is a failure in communications, the communication system will 'send' a *T-fault* message to the process for each message sent by that process which cannot be delivered with the defined semantics. This message contains the TID for the trans-

action during which the message was sent. Note that the *T-fault* message does not indicate that the message was not transmitted correctly, but only that the communication subsystem cannot verify the correct transmission.

## 2.3 Protocol States

In **STP**, a process may execute many concurrent transactions. Each transaction is in one of the following states:

### **null**

A transaction is in the **null** state until the initialization exchange is completed. In our model, this is the starting state for all transactions.

### **sending, receiving**

A transaction which has been successfully initialized and is currently active (its processes are currently exchanging messages) will be either in the **sending** or **receiving** states.

### **ready**

When a process wishes to commit a transaction, the transaction enters the **ready** state. This signifies that data transfer is over. Note that only one process (the one that tries to commit the transaction) will enter this state.

### **committed**

If both processes agree to end the transaction successfully, the transaction will enter the **committed** state, a terminal state.

### **aborted**

The other terminal state for a transaction is the **aborted** state. This signifies that a transaction failed for some reason.

## 2.4 Protocol State Transitions

Transactions move between states according to messages sent or received. Invalid messages (those without a verifiable signature or that are not acceptable in the current state of the transaction) are discarded without affecting the state of the transaction.

### **null state**

Processes begin each transaction in the **null** state. For a transaction in this state, processes may send the following message:

*T-start* will cause the transaction to remain in the **null** state.

Processes may receive the following messages:

*T-start* may be accepted or rejected. If the transaction is accepted, the transaction moves into the **receiving** state; if it is rejected, the transaction remains in the **null** state.

T-accept will cause the transaction to move into the *sending* state if the message was sent in response to a T-start message sent previously by the process.

T-fault has no effect on the process.

#### **sending state**

If a transaction is in the *sending* state, a process may send the following messages:

T-data will cause the transaction to remain in the *sending* state.

T-token will cause the transaction to move to the *receiving* state.

T-abort will cause the transaction to move to the *aborted* state.

T-commit will cause the transaction to move to the *ready* state.

Processes may receive the following message:

T-fault will cause the transaction to move to the *aborted* state.

#### **receiving state**

For a transaction in the *receiving* state, a process may receive the following messages:

T-data will cause the transaction to remain in the *receiving* state.

T-token will cause the transaction to move to the *sending* state.

T-abort will cause the transaction to move to the *aborted* state.

T-commit enables the process to commit or abort the transaction. Sending:

T-commit will cause the transaction to move to the *committed* state.

T-abort will cause the transaction to move to the *aborted* state.

T-fault will cause the transaction to move to the *aborted* state.

#### **ready state**

For a transaction in the *ready* state, a process may send no new messages and may receive the following messages:

T-abort will cause the transaction to move to the *aborted* state.

T-commit will cause the transaction to move to the *committed* state.

T-fault will cause the transaction to remain in the *ready* state and to resend T-commit.

#### **committed, aborted states**

A transaction in either the *committed* or *aborted* states is in a terminal state.

The state machine for STP, excluding the T-fault message type, is depicted in Figure 2.

## **2.5 Formal Specification**

### **Abstract Protocol Language**

Both processes execute the same protocol. A process consists of various types of data declarations and a set of guarded commands. A guarded command consists of a predicate, which serves to enable the command, and an action, which will be executed atomically. The set of guarded commands is evaluated repeatedly, and at each execution an enabled command is executed. If multiple commands are enabled, then one is selected non-deterministically, but under the following fairness condition: a command that is enabled continuously will eventually be executed.

Selection statements (**if** ...  $\square$  ...  $\square$  ... **fi**) also have the above non-deterministic and fairness properties.

One data declaration type should be mentioned: a **parameter** datum is generated each time the guarded command set is evaluated; the parameter is set to a random value of the same type as the parameter.

### **Annotations**

The STP process is passed its user identifier **self** as a process parameter and receives as input an array of identity certificates, **self.cert**. These certificates are indexed according to user identifier. A certificate is used as a parameter to the **SelfKey** function, which computes the key that the process uses to sign messages for the transaction. Messages may be signed with the **Sign** function.

Processes exchange certificates during the transaction initiation phase. A process that receives a certificate will use the **Id** and **Key** functions to extract the user identifier and the verification key from the certificate. The verification key is then used as a parameter to the **Verify** function, which validates the signature computed by the sender.

Transaction state is kept in arrays indexed according to transaction identifier. The **tid** datum is a process parameter that is reset each time the guarded command set is evaluated, and so will either be a random number or will be equal to the TID field of a received message. If the former, its uniqueness can be determined with the **Unique** function, which implicitly registers TIDs and so prevents their reuse.

The transaction state will be initialized after the transaction has been accepted, and then is updated during the subsequent exchange and termination phases. We assume that data storage is persistent and durable.

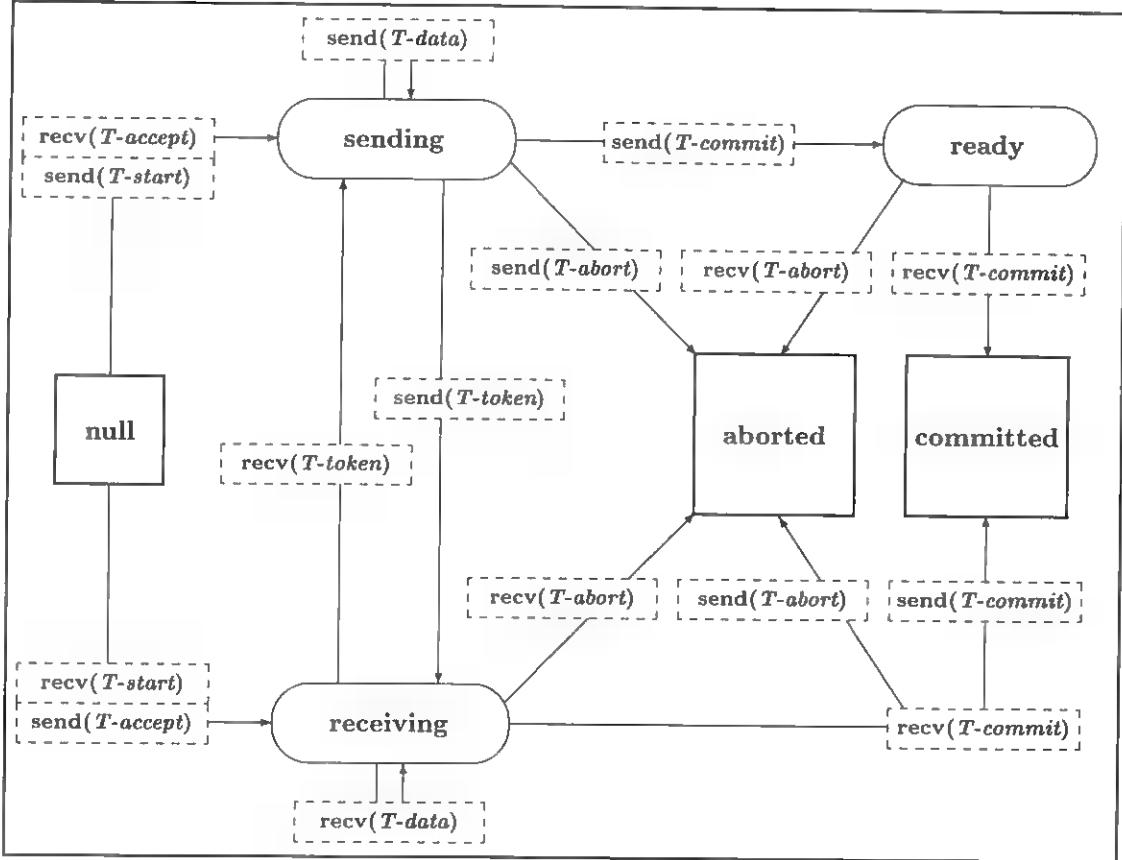


Figure 2: STP State Machine

For each transaction of which the process is a member, it will maintain the following information:

**t\_state** the current state (initially *null*)  
**t\_other** the identity of the other process  
**t\_key** the verification key  
**t\_data** the data sent or received  
**t\_seq** the current message number  
**t\_context** the current context

The context is recomputed each time data is sent or received, using the **Hash** function. This function computes a cryptographic message digest of the concatenation of its arguments.

The sections below annotate the code for each action in the process. Note that the temporary boolean variable **valid** is computed for each received message, and is true if the signature is verified and if the transaction state allows this message. Invalid messages are always dropped with no other effect.

Note as well that the signature always includes the message type, so the message body can't be used to forge a different type of message, and that the signature does not cover certificates, since these have independent integrity checks.

#### Action 0

In this section, a process attempts to begin a new transaction. If *tid* is unique and the local state for this transaction is *null*, the process will nondeterministically decide either to **skip** or to initiate the transaction.

If it decides to initiate the transaction, it selects a process (at random) along with an identity certificate corresponding to the UID of that process. It then sends a *T-start* message to the other process with the identity certificate.

#### Action 1

A process that receives a valid *T-start* message can decide to accept or reject the transaction. Rejected transactions are simply dropped - the transaction will also remain in the *null* state for the other process.

To accept the transaction, the process selects the identity certificate corresponding to the UID of the initiating process, and sends a *T-accept* message to the other process with that certificate. It then initializes the transaction variables and sets the state for the transaction to *receiving*.

## Action 2

When the transaction initiator gets a valid *T-accept* message, it verifies that it was sent by the process to whom it sent the *T-start* message. If so, it also initializes the variables for that transaction, but sets its state to *sending*.

## Action 3

A process with a transaction in the *sending* state will non-deterministically either send data, pass the token, commit the transaction or abort the transaction.

In order to send data, a process sends a *T-data* message. The current context is included with each data message to establish causality. After sending the message, the data is stored, the sequence number is updated for the next message send or receive, and the context will be updated with the new data to reflect the current state of the transaction.

When a process decides to pass the token to the other process, it sends a *T-token* message and sets the state of the transaction to *receiving*.

When a process decides to commit the transaction, it sends a *T-commit* and sets the state of the transaction to *ready*.

When a process decides to abort the transaction, it sends a *T-abort* message and sets the state of the transaction to *aborted*.

## Action 4

When a process receives a valid *T-data* message, it discards the data if the received context is incorrect; otherwise, it stores the data and updates the sequence number for the next message send or receive.

## Action 5

When a process receives a valid *T-token* message, it will abort the transaction if the received context is incorrect; otherwise it sets its state to *sending*.

## Action 6

When a process receives a valid *T-commit* message, it will process the message according to the state of the transaction.

If the transaction is already *aborted* or *committed*, the process's original *T-abort* or *T-commit* message was probably lost. and so the process resends the appropriate message.

If the transaction is in the *ready* state, then the message is in response to the process's earlier commit request, and the process sets the transaction state to *committed*.

If the transaction is in the *receiving* state, then the process non-deterministically either aborts or commits the transaction, and sets the transaction state accordingly. It will only commit the transaction if the received context matches its own, but note that it does not have to do so. If it does choose to accept the commit request, it returns a *T-commit* message that has a null data context field. Note that by accepting the commit, it implicitly asserts that its transaction context is equal to the received context, and so there is no reason to return its own value for context.

## Action 7

When a process receives a valid *T-abort* message for a transaction, it sets the state of that transaction to *aborted*.

## Action 8

When a process receives a *T-fault* message for a transaction, it processes it according to the state of that transaction. If the transaction is active (*sending* or *receiving*), the transaction is aborted. If the transaction is in the *ready* state, it is waiting for a response to its commit request, and so the process resends it. Otherwise the process does nothing.

## STP

```
process STP[self : 0..MAXUID]
constant
    MAXUID,
    MAXTID,
    MAXSEQ,
    MAXCERT,
    MAXSIGN,
    MAXHASH,
    MAXKEY
input
    self.cert : array [0..MAXUID] of 0..MAXCERT
function
    Sign(0..MAXKEY, integer) returns 0..MAXSIGN,
    Verify(0..MAXKEY, integer) returns 0..MAXSIGN,
    Hash(integer) returns 0..MAXHASH,
    Unique(0..MAXTID) returns boolean,
    Id(0..MAXCERT) returns 0..MAXUID,
    Key(0..MAXCERT) returns 0..MAXKEY,
    SelfKey(0..MAXCERT) returns 0..MAXKEY
variable
    msgtype : set { T-start, T-accept, T-data, T-token, T-commit, T-abort, T-fault },
    state : set { null, sending, receiving, ready, committed, aborted },
    t_state : array [0..MAXTID] of state init null,
    t_other : array [0..MAXTID] of 0..MAXUID,
    t_key : array [0..MAXTID] of 0..MAXKEY,
    t_data : array [0..MAXTID, 0..MAXSEQ] of integer,
    t_seq : array [0..MAXTID] of 0..MAXSEQ,
    t_context : array [0..MAXTID] of 0..MAXHASH,
    cert : 0..MAXCERT,
    data : integer,
    sign : 0..MAXSIGN,
    context : 0..MAXHASH,
    valid : boolean
parameter
    tid : 0..MAXTID
begin
    (t_state[tid] = null ∧ Unique(tid)) ⇒ {Action 0}
        t_other[tid] := any;
        cert := self.cert[t_other[tid]];
        sign := Sign(SelfKey(cert), T-start||tid||t_other[tid]);
        send T-start(tid, sign, cert) to t_other[tid];
    □ rcv T-start(tid, sign, cert) ⇒ {Action 1}
        valid := (sign = Verify(Key(cert), T-start||tid||self) ∧ t_state[tid] = null);
        if true →
            skip;
        □ (valid) →
            t_state[tid], t_seq[tid], t_context[tid] := receiving, 0, 0;
            t_other[tid], t_key[tid] := Id(cert), Key(cert);
            cert := self.cert[t_other[tid]];
            sign := Sign(SelfKey(cert), T-accept||tid);
            send T-accept(tid, sign, cert) to t_other[tid];
        fi

```

□ **rcv**  $T\text{-accept}(tid, sign, cert) \Rightarrow \{\text{Action 2}\}$   
     valid := (sign = Verify(Key(cert),  $T\text{-accept}||tid$ )  $\wedge$  t.state[tid] = null);  
     **if** ( $\sim$ valid  $\vee$   $\sim$ (Id(cert) = t.other[tid]))  $\rightarrow$   
         **skip**;  
     □ (valid  $\wedge$  Id(cert) = t.other[tid])  $\rightarrow$   
         t.state[tid], t.seq[tid], t.context[tid] := sending, 0, 0;  
         t.key[tid] := Key(cert);  
     **fi**  
 □ (t.state[tid] = sending)  $\Rightarrow \{\text{Action 3}\}$   
     cert := self.cert[t.other[tid]];  
     **if** true  $\rightarrow$   
         data := any;  
         sign := Sign(SelfKey(cert),  $T\text{-data}||tid||data||t\text{-context}[tid]$ );  
         **send**  $T\text{-data}(tid, data, t\text{-context}[tid], sign)$  to t.other[tid];  
         t.data[tid, t.seq[tid]] := data;  
         t.context[tid] := Hash(data||t.context[tid]);  
         t.seq[tid] := t.seq[tid] + 1;  
     □ true  $\rightarrow$   
         sign := Sign(SelfKey(cert),  $T\text{-token}||tid||t\text{-context}[tid]$ );  
         **send**  $T\text{-token}(tid, t\text{-context}[tid], sign)$  to t.other[tid];  
         t.state[tid] := receiving;  
     □ true  $\rightarrow$   
         sign = Sign(SelfKey(cert),  $T\text{-abort}||tid$ );  
         **send**  $T\text{-abort}(tid, sign)$  to t.other[tid];  
         t.state[tid] := aborted;  
     □ true  $\rightarrow$   
         sign = Sign(SelfKey(cert),  $T\text{-commit}||tid||t\text{-context}[tid]$ );  
         **send**  $T\text{-commit}(tid, t\text{-context}[tid], sign)$  to t.other[tid];  
         t.state[tid] := ready;  
     **fi**  
 □ **rcv**  $T\text{-data}(tid, data, context, sign) \Rightarrow \{\text{Action 4}\}$   
     valid := (sign = Verify(t.key[tid],  $T\text{-data}||tid||data||context$ )  $\wedge$  t.state[tid] = receiving);  
     **if** ( $\sim$ valid  $\vee$   $\sim$ (context = t.context[tid]))  $\rightarrow$   
         **skip**;  
     □ (valid  $\wedge$  context = t.context[tid])  $\rightarrow$   
         t.data[tid, t.seq[tid]] := data;  
         t.context[tid] := Hash(data||t.context[tid]);  
         t.seq[tid] := t.seq[tid] + 1;  
     **fi**  
 □ **rcv**  $T\text{-token}(tid, context, sign) \Rightarrow \{\text{Action 5}\}$   
     valid := (sign = Verify(t.key[tid],  $T\text{-token}||tid||context$ )  $\wedge$  t.state[tid] = receiving);  
     **if** ( $\sim$ valid)  $\rightarrow$   
         **skip**;  
     □ (valid  $\wedge$   $\sim$ (context = t.context[tid]))  $\rightarrow$   
         cert := self.cert[t.other[tid]];  
         sign := Sign(SelfKey(cert),  $T\text{-abort}||tid$ );  
         **send**  $T\text{-abort}(tid, sign)$  to t.other[tid];  
         t.state[tid] := aborted;  
     □ (valid  $\wedge$  context = t.context[tid])  $\rightarrow$   
         t.state[tid] := sending;  
     **fi**  
 □ **rcv**  $T\text{-commit}(tid, context, sign) \Rightarrow \{\text{Action 6}\}$   
     valid := (sign = Verify(t.key[tid],  $T\text{-commit}||tid||context$ )  
                $\wedge$   $\sim$ (t.state[tid] = sending  $\vee$  t.state[tid] = null));

```

if ( $\sim$ valid)  $\rightarrow$ 
    skip;
□ (valid  $\wedge$  t_state[tid] = aborted)  $\rightarrow$ 
        cert := self_cert[t_other[tid]];
        sign := Sign(SelfKey(cert), T-abort||tid);
        send T-abort(tid, sign) to t_other[tid];
□ (valid  $\wedge$  t_state[tid] = committed)  $\rightarrow$ 
        cert := self_cert[t_other[tid]];
        sign := Sign(SelfKey(cert), T-commit||tid||0);
        send T-commit(tid, 0, sign) to t_other[tid];
□ (valid  $\wedge$  t_state[tid] = ready)  $\rightarrow$ 
        t_state[tid] := committed;
□ (valid  $\wedge$  t_state[tid] = receiving)  $\rightarrow$ 
    if true  $\rightarrow$ 
        cert := self.cert[t_other[tid]];
        sign := Sign(SelfKey(cert), T-abort||tid);
        send T-abort(tid, sign) to t_other[tid];
        t_state[tid] := aborted;
    □ (context = t_context[tid])  $\rightarrow$ 
        cert := self.cert[t_other[tid]];
        sign := Sign(SelfKey(cert), T-commit||tid||0);
        send T-commit(tid, 0, sign) to t_other[tid];
        t_state[tid] := committed;
    fi
    fi
□ rcv T-abort(tid, sign)  $\Rightarrow$  {Action 7}
    valid := (sign = Verify(t.key[tid], T-abort||tid)
         $\wedge$   $\sim$ (t.state[tid] = null  $\vee$  t.state[tid] = committed));
    if ( $\sim$ valid)  $\rightarrow$ 
        skip;
    □ (valid)  $\rightarrow$ 
        t.state[tid] := aborted;
    fi
□ rcv T-fault(tid)  $\Rightarrow$  {Action 8}
    if (t.state[tid] = null  $\vee$  t.state[tid] = committed  $\vee$  t.state[tid] = aborted)  $\rightarrow$ 
        skip;
    □ (t.state[tid] = sending  $\vee$  t.state[tid] = receiving)  $\rightarrow$ 
        t.state[tid] := aborted;
    □ (t.state[tid] = ready)  $\rightarrow$ 
        cert := self.cert[t_other[tid]];
        sign := Sign(SelfKey(cert), T-commit||tid||t.context[tid]);
        send T-commit(tid, t.context[tid], sign) to t.other[tid];
    fi
end

```

### 3 Related Work

In this section, we describe several protocols currently in use or under development. All of these provide the properties that are required for secure messages; none of them, however, provide the relational properties that are required for secure transactions.

#### PGP and PEM

Pretty Good Privacy (PGP) and Privacy Enhanced Mail (PEM) are used mainly for electronic mail security. [8] They are datagram-like in that they allow users to send single, asynchronous messages to each other.

PEM is more flexible, in that it offers a choice of cryptographic algorithms and certificate authorities, although these can't be negotiated since messages are asynchronous. PEM is also more extensible, since PEM data are passed in named header fields, as specified in [1]. PGP requires less infrastructure and is thus suited for more ad hoc environments.

#### SSL and S-HTTP

Secure Sockets Layer (SSL) [2] and Secure Hypertext Transport Protocol (S-HTTP) [7] allow for the exchange of multiple messages between two processes. The main difference between these protocols and PGP and PEM is that SSL and S-HTTP use a session model, and thus the security mechanisms and parameters used during a session can be negotiated. This allows the degree and kind of security to be varied according to such factors as the nature of the data being exchanged and the vulnerabilities of the underlying communication media. SSL and S-HTTP were designed primarily for WWW-based commerce.

In terms of implementation, SSL fits between the session and transport layers, and is implemented as a replacement for the sockets API to be used by applications requiring secure communications. S-HTTP, on the other hand, is similar to PEM in terms of implementation - its data are passed in named text fields in the HTTP header.

#### SET

The Secure Electronic Transaction (SET) [11] specification was introduced this year by several credit card and system software vendors. It is currently under design and there are no publicly available reference implementations.

SET uses a session communications model like that of SSL and S-HTTP. SET certificate management is based on X.509, and uses a fixed structure essentially the same as that used for credit card distribution. Both merchants and customers obtain certificates from certificate authorities that represent one or more credit card brands. In order to transact business, a merchant and customer must use certificates obtained from a common brand.

The protocol provides an ad hoc security model that is based on credit card usage. In particular, SET transactions involve only the secure exchange of financial information - negotiations and the actual selection of merchandise are done out of band. Transactions involve a single merchant and consumer engaged in one exchange. The relationship between merchant and customer is asymmetric.

As with the other protocols, SET provides the properties required for secure messages. Note, however, that in recent drafts the specification has defined a linkage function that may be used to insert a reference to one message into another message. The purpose of this function is not completely specified but it could be used to enhance the SET protocol to provide the causality property.

### 4 Conclusion

In this paper we have presented a protocol for secure transactions that adds causality, atomicity and isolation to the secure message properties of privacy, authentication and integrity that are already provided by current protocols like SET, PGP and SSL. This allows secure transactions to be based on relational properties across messages rather than the properties of single messages.

This protocol is complementary to the current protocols, in that existing elements like identity certificates are used in a compatible manner and new elements like context are orthogonal to the other elements in these protocols. Thus, the new properties can be added easily to these protocols.

### References

- [1] D. Crocker. Standard for the format of arpa internet text messages. Technical Report RFC822, IETF, August 1982.
- [2] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The ssl protocol v3.0. Internet Draft, March 1996.

- [3] Mohamed Gouda. Elements of network protocol design. Contact author for further information.
- [4] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1993.
- [5] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network Security: Private Communication in a Public World*. Prentice Hall, Englewood Cliffs, NJ, 1995.
- [6] Larry L. Petersn and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.
- [7] E. Rescorla and A. Schiffman. The secure hypertext transfer protocol. Internet Draft, December 1994.
- [8] Bruce Schneier. *E-Mail Security: How to Keep Your Electronic Messages Private*. John Wiley and Sons, Inc., New York, NY, 1995.
- [9] Bruce Schneier. *Applied Cryptography*. John Wiley and Sons, Inc., New York, NY, second edition, 1996.
- [10] Jiawen Su and J. D. Tygar. Building blocks for atomicity in electronic commerce. In *The 6th Usenix Security Symposium*, pages 97–102. Usenix, July 1996.
- [11] VISA, MasterCard, et al. Secure electronic transaction specification, books 1-3. WEB document, June 1996.
- [12] Frank Lloyd Wright. *Modern Architecture*. Princeton University Press, Princeton, NJ, 1931.

# PayTree: “Amortized-Signature” for Flexible MicroPayments

Charanjit S. Jutla\*

Moti Yung†

## Abstract

We present the idea of *PayTree*, a method to amortize the work of a single signature production and verification among numerous micropayments in the model where payments are made from a payer to a set of merchants (under various trust assumptions regarding these merchants).

The PayTree scheme is simple yet flexible. Its main feature is that it can support arbitrary (dynamically determined) number of payees while using a single signature (unlike PayWord). It is therefore suitable for supporting users who “shop around”. The scheme is easily extendible dynamically (without the use of further signatures) and has a reasonable computational penalty. It can be viewed as a “divisible coin” mechanism as well.

## 1 Introduction

A payment system needs to be efficient, so that the cost of operation of a transaction is favorable compared to the monetary value of the transaction itself. This is the logic behind various systems and designs.

A computational bottleneck in many systems is the public-key operations like “signature” [4, 17]

which is essential in assuring the validity of payment in cash and micro-payment systems.

To solve the problem, a suggestion that spreads a signature value over many cryptographic values derived by more efficient cryptographic one-way (hash) function [12, 15] was suggested by Shamir and Rivest recently [16], and independently by various other researchers (Pedersen [14] and Anderson et al. [1].) The mechanism, known as PayWord, exploits Lamport’s idea [8] (implemented as S/key). Namely, to sign a target value which is derived by many repeated applications of a one-way function, and opening the target value one step at a time in the chain of function applications; each time the opening looks like “inverting” the one-way function.

While being useful as a micropayment mechanism between two entities (a user and a merchant), PayWord does not provide a general purpose spending mechanism to be used by a user in a multi merchant system (namely, a user who shops around in an electronic shopping center). In such an environment one prefers a signature which is made public and can be spent in many ways (especially when merchants are relatively honest but want to assure fast or cheap receipt of valid payment).

In this paper we suggest “PayTree”, a general purpose mechanism that amortizes a public-key signature scheme against spending of many

---

\*IBM T.J. Watson Research Center

†Bankers Trust Electronic Commerce/ CertCo

coins, paid to either a single or to many merchants. The mechanism employs “authentication tree” techniques, it has preprocessing similar to PayWord, and when used in a single merchant (payee) environment, it has similar cost of payment (when amortized over many micropayment— which is the typical envisioned application for the mechanism). The mechanism enables to split the signature on micro-coins among merchants (and split the coin signed), so it can be called an “amortized-signature” mechanism.

PayTree has similar security guarantees as PayWord (related model of trust and entity relationships, with some crucial changes based on multi payees as we will note), while providing better flexibility and multi payees. PayTree exploits Merkle [9] authentication tree and it also has an extendible “spending potential” using a “renewal capability” as in [10] (whereas PayWord is limited in this respect). The mechanism can be viewed as a “divisible coin” as well.

The mechanism can enhance systems which employ digital signature as a basic tool for amortization of signatures in payments and other “fixed” or “almost fixed” used-once authorization token schemes.

## 1.1 The model

Let us briefly review the “PayWord” model of Shamir and Rivest which we adopt and extend.

We assume that a user gets a signature scheme authorized (certified) by an authority (a broker) to issue “commitment to payment”, that is it can allocate a budget for billing and then spend the budget in an incremental way. Alternatively the broker may sign strings representing monetary value directly.

Bills are respected by the broker and actual money transfer from the payer to the payee is performed. A bill spent has to be both “valid” and “uniquely spent”.

The payee recognizes the certification and the signature, and has the means to recognize the “validity” of a bill it gets. The idea is to increment the bill as payments are transferred in a fast online fashion; whereas the broker is in charge of detecting “double spending”, and it respects the first one and rejects the second payment with the same coin. More details of the model and trust relationships are in [16].

### 1.1.1 Assumptions on Merchants

We classify three models regarding payees (merchants):

- The single payee model.
- The multiple non-colluding payees model.
- The multiple potentially-colluding payees model.

Here we assume that based on a single commitment (posted in public, say) various merchants or a single merchant can get paid. The single merchant model is exactly as PayWord (i.e. the signature is associated with and is given to it), while, in general, PayTree extends PayWord’s functionality. With multi payee setting, we have to make sure that we operate in an environment where merchants do not collude to present “double spending”, or (especially in the case of big payments) that we have other mechanisms to cope with it (and this is doable as will be explained in the sequel). In the multi-payee setting signature are published and are not associated with a specific

payee. The payment is associated with a receiver. Either a payment is given to a non-colluding merchant or is strongly associated with a (potentially misbehaving) merchant ID.

## 2 PayTree: the mechanism

### 2.1 The Basic PayTree

Let  $h$  be a one way function  $h : \{0,1\}^n \rightarrow \{0,1\}^n$ . Let  $g_k$  be one way functions  $g_k : \{0,1\}^{nk} \rightarrow \{0,1\}^n$ . (Both functions can be based on fast cryptographic hash functions like SHA1 or MD5). A k-ary *PayTree* is the following labeled k-ary tree structure  $\{N, r, \delta, L, h, g_k, F, R\}$ :

- $N$  is the set of nodes of the tree
- $r$  is the root of the tree
- $F \subseteq N$  is the set of leaf nodes
- $\delta$  is the successor function  $\delta : (N - F) \times [0..k-1] \rightarrow N$
- $L$  is the labeling of the nodes of the tree  $L : N \rightarrow \{0,1\}^n$
- $h$  and  $g_k$  are one-way functions as above
- $R$  is the secret labeling of the leaves  $R : N \rightarrow \{0,1\}^n$ . Thus the leaves have two labels.

In addition, the following holds for the PayTree:

1. For every node  $s \in F$ :  $L(s) = h(R(s))$
2. For every node  $s \in N - F$ :  $L(s) = g_k(L(\delta(s, 0)), L(\delta(s, 1)), \dots, L(\delta(s, k-1)))$

### 2.2 Signature

A party  $I$  wishing to pay using the PayTree, generates such a PayTree  $T$  and keeps the random numbers ( $R$ ) labeling the leaves secret.  $I$  signs the label of the root  $r$  of the tree. Let this signature be called  $S(T)$ . In case, the protocol supports different height trees, the height of the tree is also included in the information signed. We assume that the trees are balanced, i.e. each leaf is at the same depth.

In fact, building on the same scenario as PayWord, the signature can be performed by a broker (or by a signature scheme certified by it). The broker later will collect the tree back and will detect double spending. The user “withdraw money or credit” from the broker by getting a signature on the tree; it then spends the tree leaves, paying to merchants. The merchants will give the leaves (coins) back to the broker who reconstructs the tree and checks for double spending. The tree is good for a limited time and the coins have to be redeemed within this period. Within a period, the broker keeps record of payment transfer together with the redeemed coins.

We can use any public key signature, RSA, DSS, etc. [17, 11].

### 2.3 Payment:

With each tree  $T$ ,  $I$  maintains a list of merchants  $M$ , and for each merchant in  $M$ , it maintains a list of leaves  $F_i$  (where  $i$  is the index of the merchant in  $M$ ) whose  $R$  values have already been disclosed to merchant  $i$ . Also,  $I$  maintains a list  $F_{all}$  which is the union of all the  $F_i$ .

Let the nodes of the tree be numbered in inorder fashion (with the first leaf numbered one). Thus the root is numbered  $k^h$ , and the last leaf is num-

bered  $k^{h+1} - 1$  (where  $h$  is the height of the tree, and  $k$  is the arity of the tree).

To pay party  $m$ ,  $I$  discloses to  $m$  the signature  $S(T)$ , and the label of the root  $r$  (these two and even the whole  $L$  can actually be made public knowledge).  $I$  checks if  $m$  is in the list  $M$ . If so, let  $i$  be the index of  $m$  in  $M$ . Otherwise, insert  $m$  in  $M$ , and let  $i$  be the index. In the latter case, create a new empty list  $F_i$  as well.

Next,  $I$  picks the smallest numbered leaf  $f \in F$  not in  $F_{all}$ . Let  $t$  be the least common ancestor of  $F_i \cup \{f\}$  (if  $F_i$  is empty, let  $t$  be the root). Let  $p$  be the path from  $t$  to  $f$ :  $p_0 = t, p_1, \dots, p_j = f$ . Now,  $L(p_0)$  has already been disclosed to  $m$ . Moreover, the merchant has already verified that  $L(p_0)$  is indeed a valid label. Thus, to prove to  $m$  that  $R(f)$  is the correct  $R$ -label of  $f$ ,  $I$  discloses to  $m$  the secret  $R(f)$  as the  $R$ -label of  $f$ , as well as the labels  $L$  of all nodes  $q_{l+1,l'}$  ( $l'$  can range from 0 to  $k-1$ ), such that  $q_{l+1,l'}$  is a successor of some  $p_l$ , where  $0 < l < j$ , and  $q_{l+1,l'}$  is not in the path  $p$ .

Disclosure of the  $R$  value of a leaf amounts to transfer of one unit of monetary value.

$I$  updates both  $F_i$  and  $F_{all}$ .

## 2.4 Verification by merchant

The first time, the merchant  $m$  receives a PayTree  $T$ , it checks that the signature is a valid signature. Let  $i$  be its index in the list  $M$  maintained by  $I$ . By induction, assume that the  $R$  labels of all leaves in  $F_i$  have already been verified by  $m$  to match the signature of the  $T$ . In the process it has also verified the labels  $L$  of all nodes on all paths from leaves in  $F_i$  to  $r$ . In particular, it has verified the  $L$ -label of the least common ancestor  $t$  of  $F_i$  and  $f$  (the leaf received). Thus, to verify that the received  $R$ -value is the correct

$R$ -label of  $f$ ,  $m$  validates the  $L$  label of  $f$  to be  $h(R(f))$ . To this end, it verifies that  $L(p_0) = g_k(L(\delta(p_0, 0)), L(\delta(p_0, 1)), \dots, L(\delta(p_0, k-1)))$ . One of the successors of  $p_0$  is in the path  $p$ , and its  $L$  label is obtained recursively, whereas the other  $L$  labels have been received from  $I$ .

## 3 Complexity of PayTree

The cost of generating and operating a PayTree involves various factors:

1. Generating random numbers for the leaves. Assume that generating one random number costs  $C_r$ . In the PayTree, the  $R$  values can be generated using a pseudorandom generator, with a random seed. Thus, we can assume that  $C_r = C_h$  (see next item).
2. Computing the labels  $L$  of the tree. Assume that computing either  $h$  or  $g_k$  costs  $C_h$ . Actually, computing  $g_k$  may be a linear function of  $k$ . However, for small  $k$  the discussion below still holds.
3. Signing the root label. Assume that signing costs  $C_s$ .
4. Verifying the signature. Assume that this costs  $C_e$ .
5. Finding least common ancestors during each payment. For a balanced tree, this operation is negligible in cost
6. Verifying the labels  $L$  during each payment

For sake of simplicity, we assume that the tree is binary. If the height of the tree is  $h$ , then the cost of generating the tree is  $C_s + C_r + 2^h C_h + 2^{h+1} C_h$ . If there is just one merchant, who receives all

the  $2^h$  payments, then the total number of verification steps ( $h$  computations) performed by this merchant is :  $2^{h+1}$ . However, in the worst case, a single payment step may take  $(h + 1)C_h$  time. Thus, if each payment is made to a different merchant, this is the cost per payment for verification (in addition to  $C_e$ ). However, if there are fewer than  $2^h$  merchants, a single merchant will require at most  $h(2^{p+1} - 1) - 1.44 * p2^p$  one-way computations per  $2^{p+1}$  payments. Thus, for example, if  $p + 1 = h - 1$ , the amortized complexity per payment for a single merchant is at worst  $0.28 * hC_h$  plus some lower order terms. However, this worst case occurs, when the payments to this merchant are interspersed with payments to other merchants in a particular manner. If the payments to a merchant are contiguous, the performance is more in the line of a small constant (i.e. independent of  $h$ ) amortized one-way computations per payment.

To contrast with a different binary tree structure, let's consider a completely skewed binary tree. In this binary tree there is one long path of length  $2^h$ , and from each internal node of this path, there is another child node which is a leaf (that is in  $F$  as per section 2.1). Again if there is just one merchant then the total number of computations required for all the  $2^h$  payments is as in the balanced tree. However, if each payment is made to a different merchant, then on the average each merchant would require  $2^h$  computations per payment.

## 4 Variations on PayTrees

Various additions and refinements are possible. We introduce a few of them.

### 4.1 Non-binary PayTrees

It is possible to use a larger degree trees rather than binary. It is also possible to change the degree of the tree in different levels, e.g., have a balanced binary tree where each end point connects to 20 branches at its last level, say. This last variation of degree-changing tree seems useful as it enables the “tagging” of leaves.

It is also possible to use a non-tree structures and sub-structures like in one-time signature dags (direct acyclic graphs) [2]. For simplicity, we do not include this option here.

### 4.2 PayTrees with Receiver Designation

Now we assume the case (trust model) where merchants may collude. This is an important extension where large payments are involved where the advantage of PayTree is obvious.

The basic idea is to “tag” a payment with the “name” of the merchant so that the payment is non-transferable.

Let us assume that merchants names are 10-bit long. Now we can use the degree-changing tree just mentioned. In the leaf level there are 20 applications of one-way function which are treated as 10 pairs. Each pair (a window) can sign a bit by opening the preimage of the one-way function of the first value (for 0) or the second value (for 1). This is the Diffie-Lamport one-time signature scheme (see, e.g. [10]). A payment includes opening of the bits corresponding to the name of the merchants and the path in the tree to the root; this path is not to be reused. Now, in order to transfer a payment to another merchant the merchants have to forge a new one-time signature. There are other ways to “tag” a payee’s name into a PayTree.

To implement such one-time signature efficiently one can use the ideas of Winternitz reported e.g. in [9]. Let us explain the mechanism. Essentially a row of PayWords is implemented (e.g., a five element row, each element can be opened as a digit from 0 to 9), and an extra check PayWord, which can be opened as 50 different values (a chain of size which is the sum of the chains of the rows). The rows are a register. We can go forward in the opening of the PayWord in each position, each prefix designating a different digit (so we can open in one of ten ways). We can reverse the one-way function at most 50 times on the rows. If in total we opened  $k$  function applications  $k = \sum k_i$  where  $k_i$  is the value of the digit of position  $i = 1,..5$ , The extra check PayWord is opened  $50 - k$  times. This implies that changing any position in the register is impossible (the total length of opened PayWord chains is 50). Assuming the merchants are represented as a 5-digit number ( $10^5$  merchants or “hashed names” of merchants), we have designed a mechanism that enables the opening of a leaf in a merchant-customized fashion. At this point if double spending to two different merchants happens, it is assumed that the payer is responsible for that faulty behavior. Note that the cost of such a mechanism is quite reasonable for a large merchant population, when the one-way function is really cheap.

Now, under the intractability of inverting the function, double spending detected by the broker actually implies wrong-doing of the payer, since the merchants cannot forge the signature scheme.

### 4.3 PayTrees with PayWords

To improve the performance, in cases where the payment to a merchant is interspersed with payments to other merchants, it is advantageous to

have PayWord as leaves of the PayTree. If we implement the previous idea of receiver designation, we can add an initial value in addition to the “register” based on the row of initial values. The PayWord allow payments to be made to the same merchant in a further incremental fashion. However, a right balance needs to be struck as to the height of the PayWord and that of the PayTree.

### 4.4 PayTrees with multiple denominations

We first describe PayWord with multiple denominations, as this makes the exposition simpler. To achieve a multiple ( $k$  different) denomination payment mechanism (for a single merchant), one needs to build  $k$  PayWord using a one-way function  $h$ , and then their roots collapsed to a single value using  $g_k$ . This single value can then be signed and made public knowledge. It is previously decided among the three parties (bank, payer and merchant), as to what the denominations of the  $k$  different threads(PayWord) are. To pay a unit value in the  $i$ th denomination, the payer just discloses the next preimage in the  $i$ th PayWord.

In a PayTree, we could either have multiple denomination PayWord per leaf, or a single denomination PayWord per leaf (but different denominations for different leaves). The advantage of the latter is that, we can distribute the secret information to more merchants, for the same size of secret information. The disadvantage of the latter approach is that the merchant will have to verify more paths in the PayTree.

The latter approach can be made more advantageous by allowing the denomination of a leaf (and hence its PayWord) to be determined at the payment time. This is achieved by having at each leaf a set of windows (which allows the payer to

disclose the correct denomination at the payment time, by opening the appropriate windows). The random numbers required for these windows can be pseudorandom, and the payer need not store all the pseudorandom numbers. With this approach the use of the information in the PayTree is more efficient.

#### 4.5 PayTrees with unlimited payment potential

The PayWord at each leaf  $f$  can have the special property, that the  $m$ th pre-image of label  $L$  of  $f$  is  $g_k$  of  $k$  pairs of images of  $h$ , where  $k$  is the no. of bits in the range of  $h$ . Using the idea in [10] this allows for the disclosure of a new PayWord root of this type after  $m - 1$  payments. This process can continue indefinitely. Thus, after every  $m - 1$  payments, the merchant has to verify using a  $g_k$  computation. The “renewal” of a new PayTree authorized via an old PayTree can be put on a bulletin board, enabling further usage of the tree in a multi merchant environment.

This enables a flexible amortization of a public key signature.

#### 4.6 PayTree as a divisible coin mechanism

One can view PayTree as a divisible coin into linkable sub-coins [13] (which built a number theoretic tree structure, rather than our more amorphic (any one-way function based) and (thus) efficient construction). One can spend merged sub-coins by paying with a subtree node which has a value which is the sum of its leaves. Another way is to view every opening along a path from the root as opening extra value (same as in PayWord).

## 5 What has been achieved

We claim a relatively simple yet strong mechanism:

1. A basic “amortized signature” as a micropayment mechanism for billing in the single or non-colluding merchant model, and with measures against such collusion when assumed. The broker can off-line get the coins (within a designated time limit) and transfer money between accounts. The users cannot cheat unless they can invert one-way hash functions (the actual proof relies on the hash function being an ideal “random-oracle like” function).
2. A way to combine the basic “amortized signature” with further refined amortization via the PayWord mechanism. Also, a way to designate various values to various parts of the mechanism.
3. An extendible mechanism to keep on amortizing a signature as far as we want without further use of a public key.
4. A way to implement a divisible coin.

The mechanism can be embedded in a transaction system to pay bills, to pay small amounts of money (micropayments), and bigger amounts if measures against payees’ collusions are implemented. An example that can use PayTree is a preparatory mechanism to handle “giving change” [3], where small residual values are expected. In fact, the

PayTree can modularly enhance existing mechanisms (e.g. [19, 18, 7]). For example, NetBill [18] involves the server in each transaction and signatures are used heavily, so PayTree can relax this potential burden and provide a lighter

weight option. Netcash [7] is a design for payment that employs the trusted server on-line as well, but using the Kerberos model. PayTree can help as a divisible coin in such an environment. Millicent [5], on the other hand, uses symmetric keys and hash functions which makes it very “light weight.” However, it relies on “trusting the merchants” while defending against customers. Our models, in contrast, either do not trust or only partially trust the merchants, yet we need to employ a signature scheme.

We remark that assuming that signature generation (e.g. RSA) is about  $10^4$  times slower than a one-way function/hash application (based on Message Digest like hash), a tree of size 1024 makes sense, or 512 assuming extra PayWord or receiver identifier tags are added to leaves.

We comment that the properties outlined above can be proved formally, assuming an ideal hash function (which acts as a random oracle) and one-way functions. Any ability to double spend, claim more money than actually was gotten, framing of payers, etc. implies an easy inversion of the assumed hard function.

## References

- [1] R. Anderson, H. Manifavas, and C. Sutherland, “A practical electronic cash system”.
- [2] D. Bleichenbacher and U. Maurer, “Direct acyclic graphs, one way functions and digital signatures”, Proceedings of Crypto ’94, pp. 75-82.
- [3] E. Brickell, P. Gemmell, D. Kravitz, “Trustee based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change,” Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1995, pp. 457-466
- [4] W. Diffie and M. E. Hellman, “New Directions in Cryptography,” IEEE Trans. Info. Theory IT-22, Nov. 1976, pp. 644-654
- [5] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro, “The Millicent Protocol for Inexpensive Electronic Commerce” The 4-th Int. WWW conference, 1995.
- [6] S. Goldwasser, S. Micali and R. Rivest, “A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks,” SIAM Journal of Computing 17(2), April 1988, pp. 281-308
- [7] G. Medvinsky and B. C. Neuman, “Netcash: A design for practical electronic currency on the internet,” The First ACM Conference on Computer and Communications Security, Nov. 1993, pp. 102–106.
- [8] L. Lamport, Password authentication with insecure communication, CACM 24, 70–771, Nov. 1981.
- [9] R. Merkle, A certified digital signature, Crypto 89, 218–238.
- [10] M. Naor and M. Yung, Universal one-way hash functions and their cryptographic applications, STOC 89.
- [11] NIST FIPS PUB XX, “Digital Signature Standard,” National Institute of Standards and Technology, U.S. Department of Commerce, Draft, 1 Feb. 1993
- [12] NIST FIPS PUB 180: Secure Hash Standard, 1993.

- [13] T. Okamoto and K. Ohta, "Universal Electronic Cash," *Advances in Cryptology - Proceedings of Crypto '91*, 1992, pp. 324-337
- [14] T.P. Pedersen, *Electronic payments of small amounts*, DAIMI technical report, Arhus University, Aug. 95.
- [15] R. Rivest, "The MD5 Message Digest Algorithm," *RFC 1321*, Apr. 1992
- [16] R. Rivest, A. Shamir, "PayWord and MicroMint: Two simple micropayment schemes", May 1996. (Appeared in RSA Inc. *CryptoBytes*).
- [17] R. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, v. 21, n. 2, Feb 1978, pp. 120-126
- [18] M. Sirbu and J. D. Tygar, "NetBill: An Internet Commerce System Optimized for Network Delivered Services," *Compcon '95*, pp. 20-25
- [19] J. M. Tenenbaum, C. Medich, A. M. Schiffman, and W. T. Wong, "CommerceNet: Spontaneous Electronic Commerce on the Internet," *Compcon '95*, pp. 38-43



# Agora: A Minimal Distributed Protocol for Electronic Commerce

Eran Gabber and Abraham Silberschatz  
Bell Laboratories  
700 Mountain Ave.  
Murray Hill, NJ 07974

{eran, avi}@research.bell-labs.com

## Abstract

Agora<sup>1</sup> is a Web protocol for electronic commerce, which is intended to support high-volume of transactions each with low incurred cost. Agora has the following novel properties:

- *Minimal.* The incurred cost of Agora transactions is close to free Web browsing, where cost is determined by the number of messages.
- *Distributed.* Agora is fully distributed. Merchants can authenticate customers without access to a central authority. Customers with valid accounts can purchase from any merchant without any preparations (such as prior registration at the merchant or at a broker).
- *On-line arbitration.* An on-line arbiter can settle certain customer/merchant disputes.
- *Fraud control.* Agora can limit the degree of fraud to a pre-determined (low) level.

Agora is authenticated, secure and can not be repudiated. It can use regular (insecure) communication channels.

## 1 Introduction

The wide spread use of electronic commerce on the World Wide Web will require mechanisms for dealing with high volume of low-priced transactions. By low-priced transactions we mean transactions of low monetary value, which is close to or lower than the cost incurred in communicating with a bank. Thus, low-priced transactions implies that merchants can

not afford to communicate with the bank for every transaction. High transactions volume implies distributed protocols, since any centralized resource will become a bottleneck.

In our terminology, *banks* are financial institutes that manage *customer* and *merchant* accounts. Banks provide transaction processing capabilities similar to current credit card companies. Merchants are vendors that would like to sell goods electronically. Customers are users (people, robots, etc.) that would like to purchase goods electronically. In our scheme, we assume that banks are trusted entities, while customers and merchants are assumed to be non-trusted entities. In case of a dispute, a fourth type of entity, *arbiters*, are used to settle the dispute between customers and merchants. Arbiters are assumed to be trusted.

The World Wide Web requires a new class of protocols (Web commerce protocols), that are optimized for purchase of page like merchandise (e.g., Web pages and information that can be represented as Web pages). A Web commerce protocol should be comparable to free Web browsing (in terms of message overhead) in order to achieve low incurred transaction cost.

We assume that the number of messages, rather than message length, determines the overhead (and cost) of a protocol. In other words, the optimization criteria should be number of messages and not message length, provided that message length remains within some reasonable bounds. For example, in TCP or UDP protocols, the overhead of a short message is similar to a longer message that fits into the same number of IP packets.

Agora is a fully distributed commerce protocol that requires a total of four messages to complete a transaction (including transfer of goods) without any access to a central authority. When Agora is

<sup>1</sup>Agora is an ancient Greek marketplace. It is also the name of the Israeli coin of the lowest denomination.

employed as a Web commerce protocol, its messages can be piggybacked on existing HTTP messages without any additional messages. Hence, Agora is *minimal*, since it generates the same number of message as free Web browsing.

Agora is a credit based protocol; that is, a transaction involves transfer of an account identifier and not the actual funds. Merchants use account identifiers in order to get the funds from the bank.

Fully distributed credit based protocols are susceptible to certain types of fraud, since the bank is not contacted during the execution of every transaction. For example, customers may exceed their credit limit, and since the bank is not involved in every transaction, the transaction will be approved. As another example, if a thief gets hold of the identification string and the private key of a legitimate customer, the thief can go on an unrestricted electronic shopping spree. To overcome this inherent flaw, we present in Section 5 the enhanced Agora protocol, that uses a probabilistic algorithm to limit the amount of fraud to a pre-determined (low) level. The overhead of the enhanced Agora protocol is similar to the basic Agora protocol.

We implemented a prototype of Agora using standard Web tools: a Netscape browser and an NCSA server. The customer side was implemented by a Java applet, and the merchant side was implemented by a set of CGI scripts and a small database.

The remainder of the paper is organized as follows. Section 2 describes work on related electronic commerce protocols. Section 3 discusses the basic Agora protocol while Section 4 discusses its properties. Section 5 describes the enhanced Agora protocol, which can limit fraud to a known (low) level. Section 6 discusses the on-line arbitration protocol, which can settle certain customer/merchant disputes. In Section 7 we discuss several limitations of the Agora protocol. Section 8 discusses scalability issues. The paper concludes with a description of the implementation. The Appendix describes the embedding of the Agora protocol in HTTP Messages.

## 2 Related Work

Millicent [4] is a distributed, low-overhead, digital cash protocol. Its properties are similar to Agora. Millicent introduces a *scrip*, which is a digital money that is honored by a single vendor. Millicent is designed to support high volume of low-priced transactions. Millicent's overhead is similar to Agora when used for Web commerce. However, Millicent requires preparation before a purchase from a new vendor,

namely, purchasing a scrip from a broker. These preparations can involve as many as 6 messages. Another complication is handling of change. Normally, the value of the scrip is higher than the price of the goods. Thus the merchant returns the change to the user in the form of another scrip. Since this scrip is honored only by issuing merchant, the customer is forced to redeem it by the broker.

NetBill [2] is a robust electronic commerce protocol that provides atomicity (customer pays for delivered merchandise only) and anonymity via pseudonyms. However, NetBill requires 8 messages to complete a transaction, of which two are to/from the bank. Thus, NetBill is not applicable for low-priced transactions that are less expensive than the cost of contacting the bank.

The Secure Electronic Transaction (SET) protocol of Visa and MasterCard [8] is a credit based protocol, like Agora. However, SET requires communication with the bank for every transaction. That is, SET is not distributed, and its incurred cost prohibits low-priced transactions.

Digital cash protocols, such as DigiCash [1], provide total anonymity. However, these protocols were not designed for Web commerce, and lack proper arbitration (e.g., customer sent digital cash and didn't receive the merchandise).

The chrg-*http* protocol [7] is a recent Web commerce protocol for low-priced transactions. It was implemented with standard Web tools (Mosaic browser and CERN server). However, it ignores the complexity of real world digital cash and credit transactions. Instead, chrg-*http* assumes that merchants will bill customers at regular intervals. Customers must register with merchants before any purchase.

Since Agora uses digital signatures to authenticate messages, it does not require a secure communication channel, such as Netscape's Secure Sockets Layer (SSL) [3].

Agora implements an application layer security on top of HTTP. In this aspect it is similar to Secure HTTP (S-HTTP)[6].

## 3 Basic Protocol

The Agora protocol, as described in the introduction, involves four entities: banks, merchants, customers, and arbiters. Agora supports multiple banks. Each bank has a unique identifier,  $Bid$ , a private (secret) key  $K_b^s$ , and a public key  $K_b^p$ .

*Customers* and *merchants* must have an account with some bank before any sale or purchase can be

name	description	format
<i>Cid</i>	Customer ID	$Bid \parallel expiration\_date \parallel Cacct \parallel K_c^p$
<i>SCid</i>	Signed customer ID	$Cid \parallel S_b(H(Cid))$
<i>Mid</i>	Merchant ID	$Bid \parallel expiration\_date \parallel Mname \parallel Macct \parallel K_m^p$
<i>SMid</i>	Signed merchant ID	$Mid \parallel S_b(H(Mid))$

Table 1: Customer and Merchant Identification

initiated. Customer account name is denoted by  $Cacct$ , and merchant account name is denoted by  $Macct$ . Merchants are also identified by a unique public name,  $Mname$ , which can be their IP address or their domain name. Customers and merchants have private keys,  $K_c^s$  and  $K_m^s$ , respectively, and public keys  $K_c^p$  and  $K_m^p$ , respectively. Customers and merchants identify themselves by presenting signed IDs, which were obtained from the bank.  $SCid$  is a customer's signed ID, and  $SMid$  is a merchant's signed ID.

*Arbiters* are trusted entities (not necessarily the bank), that are responsible for settling certain disputes between customers and merchants. Arbiters have the authority to repudiate committed transactions. Each arbiter has a unique identifier,  $Aid$ , a private key  $K_a^s$ , and a public key  $K_a^p$ .

The Agora protocol uses digital signatures to sign messages. In the following discussion we will assume that we use a public key cryptography algorithm, such as RSA, for computing digital signatures. However, Agora can use any public key digital signature scheme with the appropriate changes of signature and verification operations.

The signature of message  $msg$  by entity  $x$  is denoted by  $S_x(msg)$ . The signature is computed by  $S_x(msg) = E_k(H(msg))$ , where  $k$  is the signer's private key  $K_x^s$ ,  $E$  is an encryption function, and  $H$  is a secure one-way hash function, such as MD5 or SHA.

Signature  $S_x(msg)$  is verified when  $H(msg)$  is equal to  $D_k(S_x(msg))$ , where  $D$  is a decryption function, and  $k$  is the signer's public key  $K_x^p$ .

We will use the following notation for message signing:

- $S_a(msg)$  Signing a message with arbiter's private key  $K_a^s$
- $S_b(msg)$  Signing a message with bank's private key  $K_b^s$
- $S_c(msg)$  Signing a message with customer's private key  $K_c^s$
- $S_m(msg)$  Signing a message with merchant's private key  $K_m^s$

### 3.1 Accounts

All customers and merchants must have an account with a bank. The bank uses a *billing period*, during which customer and merchant IDs are valid. The bank generates new IDs for customers and merchants every billing period. The IDs expire by the end of the current billing period or by the end of the next period (a grace period). In this way, the bank can force customers to pay their bills on time.

The protocol assumes that all participants change their private and public keys every billing period in order to prevent brute force attacks. The bank delivers its new public key to customers and merchants together with their new IDs.

Table 1 depicts the generation of customer and merchant IDs. The notation  $x \parallel y \parallel z$  will be used to denote the concatenation of  $x$ ,  $y$ , and  $z$ . For example, a  $Cid$  might be: `visa:0796:0123456789:abcdef9876+deadbeef`, where ":" is a field separator, bank ID is `visa`, expiration date is `0796`, account name is `0123456789`, and customer's public key is `abcdef9876+deadbeef`.

### 3.2 Transactions

An Agora transaction consists of four messages with the control flow as depicted in Figure 1.

- **M0.** Customer asks for a price quotation from the merchant.
- **M1.** Merchant replies with:

$$M1 = SMid \parallel seq \parallel price \parallel S_m(H(Mid \parallel seq \parallel price))$$

Where  $seq$  is a unique transaction ID and  $price$  is the requested price of the goods.  $Seq$  must be unique for the merchant only (not among merchants).

- **M2.** Customer verifies  $SMid$ , checks  $Mid$  expiration date, compares  $Mname$  with the expected merchant name, extracts  $K_m^p$  from  $Mid$ , and verifies the  $seq$  and  $price$ . Customer replies

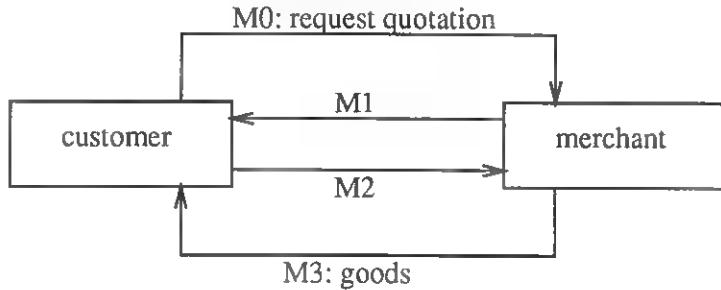


Figure 1: An Agora Transaction

with:

$$M2 = SCid \parallel seq \parallel price \parallel \\ S_c(H(Cid \parallel Mid \parallel seq \parallel price))$$

- **M3.** Merchant verifies  $SCid$ , checks  $Cid$  expiration date, extracts  $K_b^P$  from  $Cid$ , verifies signature of  $seq$  and  $price$ , and compares  $seq$  and  $price$  with  $M1$ . If  $M2$  passes all tests, then merchant commits the transaction and supplies the goods to the customer. Merchant should not charge the customer for a reload of goods that were already paid for.

Merchants submit transactions to the bank for billing by the end of the billing period. The pair  $M1$  and  $M2$  constitute a proof of the transaction.

- Message  $M0$  is a normal GET request for a menu page.
- The merchant generates  $M1$  messages for all pay-pay-view pages referenced by the menu.  $M1$  messages are embedded in the menu in such a way that they are not displayed by the browser. See an example in the Appendix.
- When the customer decides to purchase a page, he generates  $M2$  from the appropriate  $M1$  message. The subsequent GET request for the pay-per-view page includes  $M2$  as a parameter.
- The server responds with the contents of the pay-per-view page.

Note that we start a transaction and generate a  $M1$  message for each pay-per-view page mentioned in the menu. If the customer doesn't select a page, the corresponding  $M2$  messages will never be generated. A periodic garbage collection should remove old outstanding transactions.

Agora is indeed a *minimal* protocol, since it can be completely embedded in HTTP messages. Agora does not generate any additional message compared with free Web browsing. However, some messages are longer, especially menu pages that contain multiple  $M1$  messages.

## 4 Properties

The basic Agora protocol outlined in Section 3 is minimal, distributed, authenticated, and secure. We elaborate on each of these properties below.

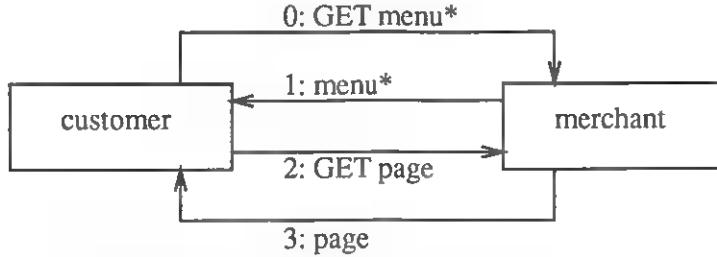
### 4.1 Minimal

A minimal Web commerce protocol should generate the same number of messages as free Web browsing. Figure 2 depicts the HTTP messages needed to access a page in the free browsing case. We will denote pages that require payment for viewing by *pay-per-view* pages. Menu pages are pages that refer to pay-per-view pages, when payment is enforced by the protocol.

Agora protocol can be piggybacked on the regular HTTP messages generated during free browsing, as depicted in Figure 3. The embedding of Agora messages in HTTP is explained below. A more detailed explanation of the embedding can be found in the Appendix.

### 4.2 Distribution

Agora is fully distributed, since both customers and merchants can verify the identity of others locally using  $K_b^P$ , the bank's public key. There is no need for access to the bank, and no preparation is needed before the transaction. Merchants need to contact the bank once in the billing period to transfer accumulated transactions.



\* a menu page contains links to other pages

Figure 2: HTTP messages for free browsing

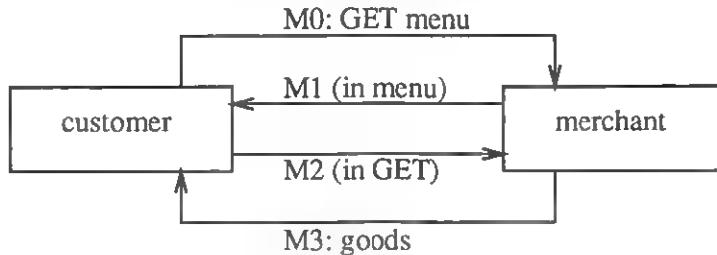


Figure 3: Piggybacking on HTTP messages

### 4.3 Authentication

Both  $M1$  and  $M2$  are signed by merchant and customer, respectively. Customers and merchants should verify signatures using  $K_m^p$  and  $K_c^p$ , which are a part of  $Mid$  and  $Cid$ , respectively. Once  $M2$  has been verified, it can not be repudiated by customer.

Customers and merchants can not create bogus accounts, since all IDs are signed by the bank's private key.

### 4.4 Security

Agora is secure against an adversary who can intercept, destroy, modify and replay messages. We now briefly outline various attacks by adversary and how to counter them.

#### • Replay

An old  $M2$  message can not be used for any other transaction by any merchant, since  $M2$  contains a unique transaction ID and the merchant's ID. Replaying  $M0$  generates a new  $M1$ . However, old  $M2$  messages never match new  $M1$ s.

#### ■ Double-Charging

Merchants can not double-charge customers, since each transaction has a unique sequence

number  $seq$ . Double charging will be detected immediately by the bank.

#### ■ Alteration

Merchants can not alter the amount charged for the goods, since the customer signed  $M2$ , which contains the original price.

#### ■ Man-in-the-Middle

The man-in-the-middle  $\mathcal{M}$  can intercept and modify all messages between customer and merchant.  $\mathcal{M}$  replaces  $M1$  with a new  $M1$  that contains a higher price quotation, his  $SMid$  and signs with his own key. Once the customer agrees to purchase,  $\mathcal{M}$  submits  $M2$  to the bank, pays the merchant for the original price and forwards the goods to the customer.  $\mathcal{M}$  pockets the difference between the inflated price and the original price.

Man-in-the-middle attack is impossible, since the customer can verify that  $M1$  contains the expected  $Mname$  in the  $SMid$  part.

### 4.5 Anonymity

Agora can be made semi-anonymous by using account aliases and not real account names in  $Cid$ . The bank should allow customers to generate new aliases for their account in every billing period. Customer should be allowed to use several aliases. The

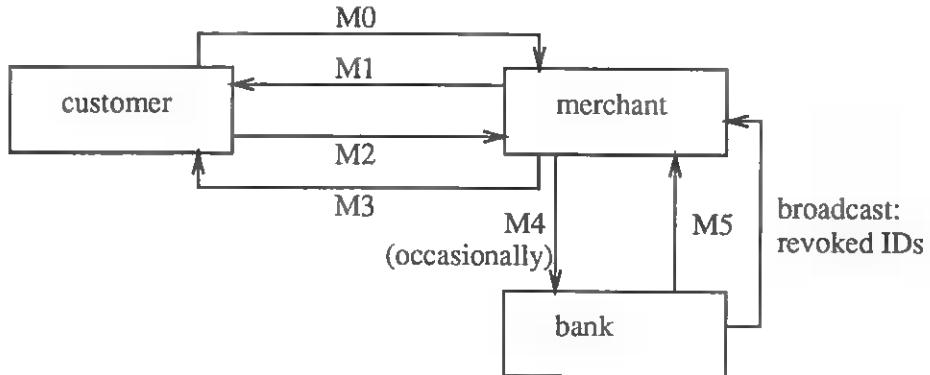


Figure 4: E-Agora

aliases should be unique, in the sense that no two customers will be allowed to use the same alias in the same billing period. Each alias should be associated with a distinct set of public and private keys. The bank will keep a translation table from aliases to real account names.

Since each customer may have several aliases, there is no single unique key that can be used for combining information about him.

Aliases provide some level of anonymity. However, full anonymity can not be achieved with current HTTP protocol, since it reveals much information to the server.

## 4.6 Untrusted Communication Channels

The protocol can use untrusted and unencrypted communication channels, since eavesdropping is not a security threat (see above), and any alteration of messages will be detected by verification of digital signatures.

## 5 Enhanced Protocol for Limiting Fraud

There are two major types of fraud that the basic Agora protocol is susceptible to: fraud by customers and fraudulent use of stolen IDs. More specific:

- Customers may exceed their credit limit by mistake or on purpose.
- Thieves may go on an electronic shopping spree with stolen  $SCids$  and  $K_c^*$ s. Since the rate of electronic purchases is very high, fraudulent purchases may create a considerable loss.

Both types of fraud are due to the distributed verification of customer IDs without access to the bank.

Credit-card companies currently allow a low level of fraud. Customer liability is limited by amount  $L$  if the customer has notified the bank within time  $T$  of the theft.

The enhanced Agora protocol denoted by E-Agora, also allows a low level of fraud. The protocol uses the following parameters:  $L$  denotes the customer liability limit,  $T$  denotes the notification period,  $R$  denotes the maximal purchase rate per period  $T$ ,  $p$  is a probability  $0 \leq p \leq 1$ , and  $M$  denotes a price threshold. Section 5.1 explains the use of  $p$  and  $M$ .

We do not consider a low rate of fraudulent purchases as a major problem, since the customer have ample time to detect it and notify the bank. Also the extent of damage is limited. A low rate is defined by purchases not exceeding threshold  $R$  per period  $T$ .

### 5.1 New Messages

The E-Agora protocol addresses the fraud problems by the following additions to the basic Agora. Merchants occasionally communicate with the bank, and the bank may broadcast a list of revoked  $Cids$  to merchants. The messages and control flow of E-Agora protocol are depicted in Figure 4.

Merchants are required to maintain a current list of revoked  $Cids$  and refuse to accept any transaction belonging to a revoked  $Cid$ . A  $Cid$  is removed from the list after its normal expiration date.

A merchant contacts the bank after verification of  $M2$  if either one of the following two conditions is true:

- The sum of current and previous purchases by the same customer exceeds a threshold  $M$ .

- A uniformly distributed random value  $r$  in the range  $[0, 1]$  is  $\leq p$ .

In all other cases, a merchant does *not* contact the bank during the transaction (as is done in basic Agora).

If any of the above two conditions is true, the merchant sends message  $M4$  to the bank:

$$M4 = Mid \parallel Cid \parallel \overline{price} \parallel S_m(Mid \parallel Cid \parallel \overline{price})$$

where  $\overline{price}$  is the sum of current and previous purchases by the same customer, which were not already sent to the bank.

After receiving message  $M4$ , the bank updates the customer balance. If the customer's purchase rate exceeds rate  $R$  per period  $T$  or the customer exceeded his credit limit, the bank does not approve the transaction and also revokes  $Cid$ . Otherwise, the bank accepts the transaction.

When processing of message  $M4$  is completed, the bank replies to the merchant with  $M5$ :

$$M5 = Cid \parallel code \parallel S_b(Cid \parallel code)$$

where  $code$  denotes whether the bank accepted or rejected the transaction.

## 5.2 Batch Processing

Merchants should communicate with the bank once in every period  $T$  and transfer batches of transactions. Usually this communication will be performed at off-peak hours. In this way, customer balance in the bank is never out of date by more than  $T$ . To reduce overhead, the bank may piggyback a list of revoked  $Cids$  on any message to the merchant.

## 5.3 Selection of Parameters

The parameters  $M$ ,  $R$  and  $p$  are derived from  $L$  and  $T$  as follows.

- $M = p \times L$
- $R = \frac{M}{T}$

The value of  $p$  is chosen so that it reduce the overhead of sending  $M4$  and  $M5$ .  $p$  should not be too small, in order to keep  $M$  reasonable large.  $M$  should represent a transaction value that is cost effective to communicate to the bank. Also,  $R$  should be an acceptable purchase rate.

As an example of a reasonable choice of parameters,  $L$  is \$50.00,  $T$  is 24 hours,  $p$  is 0.1,  $M$  is \$5.00, and  $R$  is \$5.00 per 24 hours. Since the protocol is intended for low-priced transactions, such as 0.1 cent, these limits are acceptable.

## 5.4 Discussion

A thief is a person who manages to illegally get hold of some other customer's ID and private key. A thief who go on a shopping spree will be stopped as soon as he spends  $L$  or less. A shopping spree is defined as frequent purchases in a short period of time. If the thief is purchasing expensive items (value more than  $M$ ), each item will be immediately deducted from the customer's account. The thief will be stopped after spending  $R$  during period  $T$ . If the thief is purchasing from a single merchant, even cheap items, he will also be detected as soon as he spends more than  $R$ . If the thief is buying less than  $M$  from multiple merchants, his purchases will be reported to the bank with probability  $p$ . He will be stopped as soon as his reported purchase rate exceeds  $R$ , which means that  $M$  purchases were reported to the bank. This is equivalent to total purchases of  $L$ .

A customer who exceeds his credit limit will be handled in a similar way. Since account balance is updated every period  $T$  by batch processing, the bank can immediately detect over the limit purchases of expensive items (more than  $M$ ). If the customer purchase cheap items from many merchants, he will be detected with probability  $p$  by a merchant, which means that he will make  $\frac{1}{p}$  purchases on the average before being detected. The maximal value of over the limit purchases is  $\frac{M}{p} = L$ .

## 6 On-Line Arbitration

Certain disputes may occur in the scheme we have outlined above. For example, a customer may send message  $M2$ , but receive partial goods or different goods than ordered. The customer may even not receive the goods at all. In this case the customer would like to get the goods or get a refund, since he already agreed to pay. A trusted arbiter can settle the above disputes. The arbiter can not settle more complicated disputes, such as false advertising by merchants. These disputes should be settled by humans. Figure 5 depicts the arbitration protocol and its control flow.

The arbitration protocol makes the following modification to E-Agora protocol:

- The merchant includes a secure hash value of the goods in  $M1$ .
- The customer sends  $M2$  to the merchant and expects the goods. If the customer doesn't receive the goods at all, or if the checksum of the received goods is different than what appeared

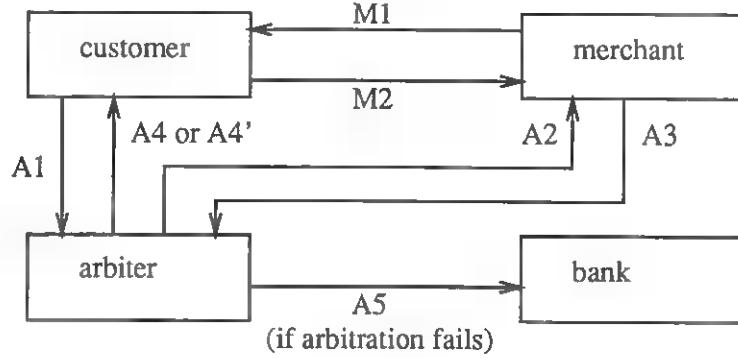


Figure 5: Arbitration Protocol

in  $M1$ , then the customer contacts the arbiter by sending message  $A1$ , where  $A1 = M1 \parallel M2$ .

- The arbiter asks the merchant for the goods by sending message  $A2$ . The arbiter signs the request, so the merchant can validate it. This prevents eavesdroppers from asking for goods that somebody else paid for.
- If the merchant replies with the correct goods in message  $A3$ , then the arbiter forwards it to the customer in message  $A4$ .
- In all other cases (merchant doesn't reply or replies with incorrect goods) the arbiter repudiates the transaction. The arbiter sends a repudiation message  $A5$  to the bank with a copy to the customer ( $A4'$ ).

## 7 Caveats

The Agora arbitration protocol assumes that the goods are static, that is, their hash value do not change from the time  $M1$  is sent to the customer until the goods are actually delivered. Another problem is that the arbitration protocol can not detect delivery delays of time-sensitive information, such as stock quotes. This means that the arbitration can not deal with dynamic or time-sensitive information.

Agora security is based on the security of the bank's signature. A security breach at the bank will allow an intruder to sign fraudulent IDs and confirm the resulting transactions. Since all other protocols assume that the bank can be trusted, we will also do so.

Agora generates new  $SCid$  for all customers every billing period. We have to use some highly secure protocol to transfer  $SCid$  and  $K_b^p$ .

Another problem is stealing  $SCid$  and  $K_b^p$  from the customer. This problem exists in all other protocols too.

## 8 Scalability

Any practical electronic commerce protocol must handle a large number of merchants, customers and transactions. We assume that the total number of active accounts is  $10^9$ , and 1% of all accounts are revoked every billing period ( $10^7$  accounts). Most revocations are due to lost or misplaced account identifiers.

The main implementation complexity of the E-Agora protocol is that all merchants must maintain a current list of revoked  $Cids$ , and refuse to accept any transaction belonging to a revoked  $Cid$ . The bank can efficiently broadcast a list of revoked  $Cids$  using some Internet broadcast protocol. Merchants can easily store a list of  $10^7$  revoked  $Cids$  in their local disk, since this list should not consume more than 100MB per billing period. The list is cleaned by the end of the billing period.

Checking a  $Cid$  against the list of revoked  $Cids$  can be performed by an application of a Bloom filter [5]. A Bloom filter consists of  $m$  bits of memory initialized to zero and  $k$  hash transformations on  $Cid$ . All hash transformation produce an integer in the range  $[0, m - 1]$ . Each revoked  $Cid$  is stored in the local disk, and the  $k$  hash transformations are applied on it. The corresponding bits of the filter are set to 1.

To check a  $Cid$  against the revoked list, the  $k$  hash transformations are computed. If any of the corresponding bits are *not* set, this  $Cid$  has not been revoked. If all corresponding bits in the filter are set, we have to check the local revoked list to ensure that this  $Cid$  has been indeed revoked, since there

is a possibility of a *filter error*. Filter errors occur when the  $k$  bits have been set by a collision with other *Cids*.

The following table depicts the probability of filter errors for representative  $m$  and  $k$ . In all cases, we assume that the filter contains  $10^7$  revoked *Cids*. For computation of filter error probability, see [5].

$m$	$k$	prob.(filter error)
$2^{26}$ bits (8MB)	4	0.040648
$2^{27}$ bits (16MB)	8	0.001652
$2^{28}$ bits (32MB)	8	0.000019

Since the entire Bloom filter can be stored in main memory, we can check a given *Cid* against the local revoked list in a constant time. I/O operations are required only for revoked IDs and infrequent filter errors.

## 9 Implementation

We implemented the Agora protocol using standard Web tools: a Netscape Navigator 2.0 browser and a NCSA 1.4.1 server. The customer side is implemented by a Java applet, and the merchant side is implemented by a set of CGI scripts and a small data base. There was no need to modify either the browser or server software. We also implemented a bank of limited capabilities, that can only generate signed IDs. The current version of the bank does not respond to *M4* messages, neither it can revoke IDs. We did not implement an arbiter yet.

The implementation is straight-forward, except for the Java applet, which had a security problem and a performance problem.

A secure Java environment prevents applets from accessing local files if the applet is loaded from the network. That is, an applet can not keep local state on the customer machine. However, for our scheme, we need to keep some local state, such as balance and a transaction log in the local machine. We evaded this problem by writing a stateless applet, that contained hard coded *SCid* and  $K_c^*$ .

Other known security problems with Netscape Navigator may allow an intruder to eavesdrop and then switch pages of the browser code or any program that the browser executes. The solution of this problem is outside the scope of the Agora protocol.

We implemented public key encryption entirely in Java (without native code), which is about 200 times slower than native C implementation. The performance will improve with JIT (Just In Time) compiler technology.

## 10 Conclusions

Agora is a simple, credit-based, low-overhead protocol for electronic commerce. The protocol is ideally suited for high-volume low-priced transactions, such as pay-per-view access to Web pages. The protocol can be piggybacked on current HTTP messages.

## References

- [1] David Chaum. Achieving Electronic Privacy. *Scientific American*, pages 96–101, August 1992.
- [2] Benjamin Cox, J. D. Tygar, and Marvin Sirbu. NetBill Security and Transaction Protocol. In *First USENIX Workshop on Electronic Commerce*, New York, New York, July 11–12 1995.
- [3] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL Protocol, Version 3.0. Internet draft ["><draft-freier-ssl-version3-01.txt>](http://draft-freier-ssl-version3-01.txt), March 1996. Found at <http://home.netscape.com/eng/ssl3/>.
- [4] Mark S. Manasse. The Millicent Protocols for Electronic Commerce. In *First USENIX Workshop on Electronic Commerce*, New York, New York, July 11–12 1995.
- [5] M. V. Ramakrishna. Practical Performance of Bloom Filters and Parallel Free-Text Searching. *Communications of the ACM*, 32(10):1237–1239, October 1989.
- [6] E. Rescorla and A. Schiffman. The Secure HyperText Transfer Protocol. Internet draft ["><draft-ietf-wts-shttp-00.txt>](http://draft-ietf-wts-shttp-00.txt), July 1995. ⇒ <http://www.eit.com/creations/s-https/>.
- [7] Lei Tang and Steven Low. Chrg-HTTP: A Tool for Micropayments on the World Wide Web. In *Proceedings of the 6th USENIX Security Symposium*, San Jose, California, July 22–25 1996.
- [8] Visa and MasterCard. Secure Electronic Transaction (SET) Specification, Book 2: Technical Specifications. Found at <http://www.visa.com/cgi-bin/vee/sf/set/settech.html?2+0>, February 1996.

## Appendix

### Embedding Agora Protocol in HTTP Messages

- Message  $M0$  is a normal GET request for a menu page.
- The merchant generates an applet activation for each reference to a pay-pay-view page in the menu. Instead of generating one  $M1$  message for each reference in the page, the merchant generates a single  $M1'$  message, which contains a concatenation of all transaction IDs and prices of the corresponding  $M1$  messages.  $M1'$  is constructed as follows (compare with Section 3.2):

$$M1' = S Mid \parallel seq_1 \parallel price_1 \parallel \dots \parallel seq_n \parallel price_n \parallel \\ S_m(H(Mid \parallel seq_1 \parallel price_1 \parallel \dots \parallel seq_n \parallel price_n))$$

Replacing multiple  $M1$ s with a single  $M1'$  reduces the number of digital signatures without compromising the security of Agora.

For example, a menu page contains the following references:

```
A picture of
<a href="protected/Everest.mpg" price="0.001">Mount Everest</a>.
A picture of
<a href="protected/ufo.mpg" price="0.001">
a UFO over the White House</a>.
```

The resulting page contains the following applet activations:

```
A picture of
<applet code="buyer.class" width=xxx height=xxx name="1">
<param name="m1" value=M1'>
<param name="nr_items" value="2">
<param text="Mount Everest">
.
</applet>
A picture of
<applet code="buyer.class" width=xxx height=xxx name="2">
<param text="a UFO over the White House">
</applet>
.
```

The applet name is its sequence number in the page. The lengthy  $M1'$  message is given only to the first applet, which sends it to all other applets. Each applet extracts its own transaction ID and price from  $M1'$ .

The resulting menu page is longer than the original page, since it contains several applet activations and one  $M1'$  message instead of direct links to pages. However, the resulting page is still contained in a single message.

- When the customer decides to purchase a page, he generates  $M2$  from the appropriate parts of  $M1'$  message. The subsequent GET request for the pay-per-view page includes  $M2$  as a parameter. For example:

```
GET http://server.address/cgi-bin/gatekeeper?M2
```

- The server responds with the contents of the pay-per-view page.

# Organizing Electronic Services into Security Taxonomies

Sean W. Smith

*IBM T.J. Watson Research Center  
P.O. Box 704  
Yorktown Heights, NY 10598  
sean@watson.ibm.com*

Paul S. Pedersen

*Los Alamos National Laboratory  
Mail Stop B265  
Los Alamos, NM 87545  
pedersen@lanl.gov*

## Abstract

With increasing numbers of commercial and government services being considered for electronic delivery, effective vulnerability analysis will become increasingly critical. Organizing sets of proposed electronic services into security taxonomies will be a key part of this work. However, brute force enumeration of services and risks is inefficient, and ad hoc methods require re-invention with each new set of services. Furthermore, both such approaches fail to communicate effectively the tradeoffs between vulnerabilities and features in a set of electronic services, and fail to scale to large sets of services. From our experience advising players considering electronic delivery, we have developed a general, systematic, and scalable methodology that addresses these concerns. In this paper, we present this methodology, and apply it to the example of electronic services offered via kiosks (since kiosk systems are representative of a wide range of security issues in electronic commerce).

## 1. The Problem

As business—commercial services provided to customers as well as government services provided to citizens—migrates to electronic settings, the contributions of security research are many: from developing underlying technology, to applying this technology to construct particular methods for secure service delivery, to verifying (both formally and experimentally) the security of these methods.

However, between the decision to enter the electronic marketplace and the decision to deploy a specific service via a specific delivery method lies a period of exploration.

---

This research was performed while the first author was with Los Alamos National Laboratory, and this paper is registered as a Los Alamos Unclassified Release. This research was sponsored in part by the Department of Energy, under contract number W-7405-ENG-36. The views and conclusions contained in this document are those of the authors alone.

We believe research needs to address this gap: how to illuminate the issues and tradeoffs between variations of services and delivery technologies. We offer this paper as an initial contribution: an attempt to extract a systematic methodology from our work in this area.

In general, vulnerability analysis consists of specifying the vulnerabilities and points of attack to which a given service is susceptible. However, in practice, we have found that managers considering electronic delivery usually provide a large set of *related* services, and implicitly expect the analysis to communicate the tradeoffs between vulnerabilities and features. Consequently, the ability to organize a set of services into a structure that clearly and concisely expresses their relations and their security risks is crucial for such an analysis to be effective. The understanding that an effective vulnerability analysis provides is, in turn, crucial for the decision on deploying electronic services to be sound.

The question, then, is how to construct and express this structure. Brute force enumeration of services and risks is inefficient and fails to meet the customer's implicit goals. Ad hoc methods require re-invention with each new set of services, and can fail to communicate the tradeoffs effectively. Furthermore, these methods do not scale to large sets of services—and communicating *anything* effectively for a large set requires understanding and exploiting organization inherent in that set.

On the other hand, a systematic methodology provides both efficiency for the analyst as well as effective communication for the customer. Furthermore, a sufficiently general methodology can also apply directly to organizing points of attack (since these are essentially just potential services unintentionally offered to the adversary). Additionally, a sufficiently expressive methodology can provide an understanding of the structure underlying a given set of services, which in turn can reveal when that set is incomplete.

From our experience in organizing electronic services into security taxonomies as part of vulnerability analyses, we have developed a general, systematic methodol-

ogy that uses the structure of a service set to organize and enumerate the corresponding vulnerability set. This methodology offers many advantages over ad hoc approaches, including making it easier to propagate changes in services to changes in vulnerabilities, to construct a complete list of vulnerabilities, to construct structured countermeasures, and to bring conciseness to service and vulnerability specifications (in some cases, reducing the length from exponential to polynomial).

Section 2 gives a theoretical presentation of the methodology. Section 3 applies the methodology to real-world examples from the domain of kiosk systems. We chose kiosk systems, since they are representative of a wide range of security issues in electronic commerce, from point-of-sale devices to networks to remote servers, from tamper-hardened public machines running dedicated browsers accessing a set of pages residing entirely on local CD-ROM to off-the-shelf private machines running Web browsers accessing the entire Web.

## 2. The Methodology

Section 2.1 establishes definitions for the terms we use in the paper. Section 2.2 considers the challenges of effectively enumerating sets of implementations and vulnerabilities. Section 2.3 presents mathematical structure organizing services into a partial order, and considers the implications of this order for vulnerability sets. Section 2.4 uses this structure as a foundation for a systematic methodology for constructing security taxonomies.

### 2.1. Definitions

In this paper, we build a mathematical structure that models real-world concepts. Since many such structures are possible, and since real-world terminology tends to be inexact anyway, we begin by defining real-world terms in the context of our model.

**Definition 1** An electronic *service* is an offered action characterized by some specific goal (which is usually defined loosely and imprecisely) and carried out via some computational device.

Examples of electronic services include transferring funds between savings and checking accounts via Automatic Teller Machines, and filing a 1040 tax form via the World Wide Web.

Typically, designers characterize a service by explicitly listing some correctness properties, but leaving many others unstated. Usually these properties divide into a

small primary class (characterizing the basic functionality of the service) and a larger secondary class (characterizing all the extra conditions necessary for the service to be correct, fault-tolerant, secure, etc.). Fully enumerating the correctness properties that a service is implicitly expected to satisfy is often a substantial challenge in security analysis. For example, designers of a certain Automatic Teller Machine system failed to make explicit the assumption that no customer would ever initiate a “withdraw cash” transaction but leave the cash in the machine [4]. As a consequence, clients engaging in this unanticipated behavior forced machines into bizarre failure modes.

**Definition 2** An *implementation* is a collection of hardware and software that realizes a particular service. In particular, an implementation must satisfy the primary correctness properties of a service. (However, in order to be fully correct, secure, and fault-tolerant, an implementation must satisfy all the properties.) For a service  $s$ , define  $\text{IMP}(s)$  to be the set of implementations of service  $s$ .

We assume that “implementations” are specified with granularity sufficiently coarse that two services can share an implementation. For example, if services  $s_1$  and  $s_2$  truly differ, than their implementations must differ at some close level of inspection—perhaps the code is not identical. However, such close scrutiny would prevent discussion of facts such as “a kiosk with sufficient database access to implement change-of-address can also implement a guide to local stores”—hence our assumption.

**Definition 3** Let  $s$  be a service. For an implementation  $I$  of this service, a *vulnerability* of  $I$  is a potential action that causes some of the correctness properties of the service to fail to hold. A vulnerability of  $s$  is the set of vulnerabilities common to all implementations of  $s$ . Define  $\text{VULN}(I)$  to be set of vulnerabilities of implementation  $I$ , and  $\text{VULN}(s)$  to be the set of vulnerabilities of service  $s$ .

### 2.2. Concisely Specifying Sets

One of the motivations for building taxonomies is to increase the effectiveness of communication to the customer. However, a simple enumeration of the elements of a set (especially a large set) is not a good example of effective communication. Consequently, we need ways to concisely describe sets such as  $\text{IMP}(s)$  and  $\text{VULN}(s)$  for a specific service  $s$ .

**Implementations** The set of implementations that realize a service is open-ended: if  $I$  realizes service  $s$ , then  $I$  along with an implementation of a completely unrelated service also realizes  $s$ . For most services that we consider, however, the implementations can be characterized as all those hardware/software combinations meeting some minimal set of requirements. Usually these requirements are implied by the customer’s specification of the service: e.g., our customer does not just want to provide change-of-address, but change-of-address via kiosks. In these situations, we can describe  $\text{IMP}(s)$  simply by listing the minimal hardware and software requirements; the fact that a kiosk with appropriate database privileges *and* the ability to play digitized excerpts from the works of Beethoven comprises an implementation of a change-of-address service is not only irrelevant, but also directly implied by saying that a kiosk with appropriate database privileges comprises an implementation.

**Vulnerabilities** The structure of the set of vulnerabilities is potentially much more complex. For example, suppose  $A$  and  $B$  are both members of some  $\text{VULN}(s)$  or some  $\text{VULN}(I)$ .

- Should “ $A$  followed by  $B$ ” also be a member? What if  $A$  is “blow up the kiosk?”
- Should “ $A$  and  $B$  concurrently” be a member?
- If  $A'$  is not a member, should “ $A'$  followed by  $B$ ” be a member? What if  $A'$  enables  $B$ ?

We provide one sample solution to this problem. First, we define compound actions.

**Definition 4** For actions  $A$  and  $B$ , define  $[A; B]$  to be the compound action consisting of doing  $A$  first, then doing  $B$ .

We now examine when such compound actions can be vulnerabilities. We consider the system consisting of implementation plus attacker to be in a particular state. With each action  $A$  we can associate two items:

- A set  $\text{PRE}_A$  consisting of all possible states of the system in which  $A$  can be applied. (As with implementation sets, this set might be expressed succinctly as a list of conditions which a state must satisfy.)
- A function  $\text{FUN}_A$  that indicates how  $A$  transforms systems states.

For actions  $A$  and  $B$ , the compound action  $[A; B]$  would be possible if  $B$  is enabled after performing  $A$ , and would have the result of doing  $A$ , then doing  $B$ . This allows us to specify the precondition set and transformation rule for  $[A; B]$ :

$$\bullet \text{ } \text{PRE}_{[A; B]} \equiv \text{FUN}_A^{-1}(\text{PRE}_B)$$

$$\bullet \text{ } \text{FUN}_{[A; B]} \equiv \text{FUN}_B \circ \text{FUN}_A$$

This calculus lets us build a closure rule, specifying when compound actions can be vulnerabilities:

- Suppose for  $B \in \text{VULN}(s)$ , action  $A$  satisfies  $\text{FUN}_A^{-1}(\text{PRE}_B) \neq \emptyset$ . Then  $[A; B] \in \text{VULN}(s)$ .

Enumerating some  $\text{VULN}(s)$  or  $\text{VULN}(I)$  reduces to enumerating some small set of atomic attacks, and then specifying the closure rule or rules which expand the atomic set into the full set.

This sample solution ignores the challenge of attacks composed of concurrent actions. Indeed, the problem of specifying all possible action combinations that comprise attacks is essentially the same as specifying all possible behaviors of computational devices (potentially with parallelism and randomization)—which is a problem that has received much attention in its own right (e.g., [8]). This solution also ignores the challenges raised by complex correctness properties—e.g., if a service must satisfy “ $X$  or  $Y$ ,” then an allowable sequence of two actions that subvert  $X$  and  $Y$  respectively may be a vulnerability (unless the second action re-enables  $X$ ). Fully exploring these problems lies beyond the scope of this paper; we will use our simple solution, but note that a suite of more advanced solutions may apply.

## 2.3. Vulnerabilities and the Service Order

### 2.3.1. The Service Order

We can use the implementation sets to order the services according to relative power. That is, one service precedes another when any implementation of the latter also supports the former; the latter is strictly more powerful.

**Definition 5** For services  $s_1$  and  $s_2$ , define  $s_1 \leq s_2$  when  $\text{IMP}(s_2) \subseteq \text{IMP}(s_1)$ . Define  $s_1 < s_2$  when this containment is proper.

This relation is a *partial order*: It is *irreflexive*: for no service  $s_1$  can it be the case that  $s_1 < s_1$ . It is *transitive*: for services  $s_1, s_2, s_3$ , if  $s_1 < s_2$  and  $s_2 < s_3$  then  $s_1 < s_3$ . (Partial orders and lattices show up in previous security work—e.g., [2, 3, 7]—but in different contexts from here.)

We can represent this structure by a directed acyclic graph:

**Definition 6** For a finite set  $S$  and partial order  $<$ , define  $\mathcal{G}(S, <)$  to be the directed graph consisting of:

- for each  $s \in S$ , a node  $\mathcal{N}_s$ ;

- for all  $s_1, s_2 \in S$ , an edge  $\mathcal{N}_{s_1} \rightarrow \mathcal{N}_{s_2}$  exactly when  $s_1 < s_2$ .

The graph  $\mathcal{G}(S, <)$  is transitively closed, since the order  $<$  is transitive. However, this graph is highly redundant: if edges exist from  $\mathcal{N}_{s_1}$  to  $\mathcal{N}_{s_2}$  and from  $\mathcal{N}_{s_2}$  to  $\mathcal{N}_{s_3}$ , then an edge also exists from  $\mathcal{N}_{s_1}$  to  $\mathcal{N}_{s_3}$ , even though the first two edges comprise a path from  $\mathcal{N}_{s_1}$  to  $\mathcal{N}_{s_3}$ . It is easily shown that such a graph has a unique *transitive reduction*, where all redundant edges — such as the one in this example from  $\mathcal{N}_{s_1}$  to  $\mathcal{N}_{s_3}$  — are removed. This reduced graph still reflects the original order:  $s_1 < s_2$  exactly when a path exists from  $\mathcal{N}_{s_1}$  to  $\mathcal{N}_{s_2}$ .

**Definition 7** For a finite set  $S$  and partial order  $<$ , define  $\widehat{\mathcal{G}}(S, <)$  to be the directed graph consisting of:

- for each  $s \in S$ , a node  $\mathcal{N}_s$ ;
- for all  $s_1, s_2 \in S$ , an edge  $\mathcal{N}_{s_1} \rightarrow \mathcal{N}_{s_2}$  exactly when
  - $s_1 < s_2$
  - for no  $s_3 \in S$  does  $s_1 < s_3 < s_2$

The edges that remain in  $\widehat{\mathcal{G}}(S, <)$  represent the *quantum* steps in the service set. We introduce notation for these steps, and for the differences that separate a weaker service from a stronger one.

**Definition 8** For  $s_1, s_2 \in S$ , define  $s_1 \xrightarrow{\delta} s_2$  when  $\mathcal{N}_{s_1} \rightarrow \mathcal{N}_{s_2}$  in  $\widehat{\mathcal{G}}(S, <)$ .

**Definition 9** For any  $s_1 < s_2$ , define  $\text{DIFF}(s_1, s_2)$  to be the differences between  $s_2$  and  $s_1$ . If the implementation sets are specified via a requirement list, then  $\text{DIFF}(s_1, s_2)$  would just be the set difference between the requirements for  $\text{IMP}(s_2)$  and those for  $\text{IMP}(s_1)$ .

**Definition 10** We write  $s_1 \xrightarrow{\delta} s_2$  when  $s_1 \rightarrow s_2$  and  $\delta = \text{DIFF}(s_1, s_2)$ .

### 2.3.2. The Structure of the Service Order

Specifying and manipulating the quantum steps that separate services depends highly on the context of the particular service set. However, in each particular context, considering the differences that characterize quantum steps raises two questions:

- Are the individual quantum steps indeed quantum?
  - Can there exist sub-quanta? If  $s_1 \xrightarrow{\delta} s_2$ , can we split  $\delta$  into  $\delta_1$  and  $\delta_2$ , and then introduce a new service  $s'$  with  $s_1 \xrightarrow{\delta_1} s' \xrightarrow{\delta_2} s_2$ ?

- Is the “quantum” step in fact a continuum? Is the continuum even a total order?

- Are the quantum steps independent? Suppose:

$$s_1 \xrightarrow{\delta_1} s_2 \xrightarrow{\delta_2} s_3$$

Can there exist services that accumulate these steps in a different order? For example, consider the possibility of an  $s'$  such that:

$$s_1 \xrightarrow{\delta_2} s' \xrightarrow{\delta_1} s_3$$

In general, should straight lines be lattices? That is, suppose  $\widehat{\mathcal{G}}(S, <)$  has a straight-line structure:

$$s_1 \xrightarrow{\delta_1} s_2 \xrightarrow{\delta_2} \dots \xrightarrow{\delta_k} s_{k+1}$$

Often the rest of the lattice has been overlooked: the graph should really consist of all  $s_P$  such that:

- $P \subseteq \{1, \dots, k\}$  (with  $s_\emptyset = s_1$  and  $s_{\{1, \dots, k\}} = s_{k+1}$ )
- $s_P \xrightarrow{\delta_i} s_{P'}$  (when  $i \notin P$  and  $P' = P \cup \{i\}$ )

### 2.3.3. Vulnerabilities and Ordering

The ordering on services leads to two rules on vulnerabilities. First, vulnerabilities move up in the order:

**Theorem 1** Let  $s_1$  and  $s_2$  be services. If  $s_1 < s_2$ , then  $\text{VULN}(s_1) \subseteq \text{VULN}(s_2)$ .

Second, vulnerabilities—coupled with an attack that removes the difference between services—move down in the order:

**Theorem 2** Let  $s_1$  and  $s_2$  be services, with  $s_1 < s_2$ . If  $B \in \text{VULN}(s_2)$  and action  $A$  subverts  $\text{DIFF}(s_1, s_2)$ , then  $[A; B] \in \text{VULN}(s_1)$ .

We also note the closure rule from Section 2.2 characterizing an implicit closure property for vulnerability sets:

**Closure Rule** If  $B \in \text{VULN}(s)$  and action  $A$  satisfies  $\text{FUN}_A^{-1}(\text{PRE}_B) \neq \emptyset$ , then  $[A; B] \in \text{VULN}(s)$ .

## 2.4. A Systematic Methodology

The theoretical structure from Section 2.3.1 provides the foundation for a systematic approach to organizing services into security taxonomies.

We begin with a set  $S$  of proposed services. We first consider construction of the service hierarchy:

- First, specify, for each  $s \in S$ , the hardware and software configurations characterizing the implementations  $\text{IMP}(s)$ .
- Use these implementation sets  $\text{IMP}(s)$  to order the services (e.g., to construct the transitively closed directed acyclic graph  $\mathcal{G}(S, <)$ ).
- Reduce  $\mathcal{G}(S, <)$  to its transitive reduction  $\widehat{\mathcal{G}}(S, <)$ , specifying the quantum steps in the service order.
- For each quantum step  $s_1 \xrightarrow{\delta} s_2$  in this service order, determine the  $\delta$  that separates  $s_1$  from  $s_2$ .
- Examine these quantum steps for completeness:
  - Is the step really quantum? If it can be further subdivided, then do so. If it is a continuum, then decide on a discrete approximation—and explicitly indicate that this is only an approximation.
  - Are there sets of independent steps? If so, complete the lattice by adding any necessary new services.

Having constructed the service hierarchy, we next consider the vulnerabilities. We note that  $\widehat{\mathcal{G}}(S, <)$  has minimal services (that is, ■ service  $s_2$  such that no  $s_1$  exists with  $s_1 < s_2$ )—these are just the sources of the directed graph.

- For each minimal service  $s$  in  $\widehat{\mathcal{G}}(S, <)$ , enumerate the basic vulnerabilities of  $s$ .
- Suppose  $s_1 \xrightarrow{\delta} s_2$  and we have enumerated the basic vulnerabilities of  $s_1$ . We then enumerate the additional vulnerabilities of  $s_2$ . (Depending on  $S$  and  $\delta$ , these may be simply be the additional risks associated with the capabilities enabled by  $\delta$ ).
- We then note that the complete set of vulnerabilities  $\text{VULN}(s)$  consists of the basic vulnerabilities, along with:
  - $\text{VULN}(s_1)$ , for all  $s_1 < s$  (from Theorem 1)
  - $[A; B]$ , for all  $B \in \text{VULN}(s_2)$ ,  $s \leq s_2$ , and suitable  $A$  (from Theorem 2 and the Closure Rule).

### 3. Examples

In this section, we apply our methodology to electronic services provided via kiosks, since kiosk systems are representative of ■ wide range of security issues, from point-of-sale devices to networks to remote servers. Section 3.1

organizes the space of services and Section 3.2 enumerates the vulnerabilities. Section 3.3 then applies this methodology to potential points of attack.

We note that, although modified for this presentation, these examples are not artificial; they not only originated from customer-directed research (e.g., [1, 5, 6]), but also demonstrated by their initially unorganized format the need for ■ systematic methodology for security taxonomies.

#### 3.1. Kiosk Services

To quote our earlier work [5], a kiosk is “a computer unit (usually publicly accessible) that provides information and services to authorized clients (usually not experienced with computers).” For customers interested in deploying services via kiosks, we undertook to enumerate the potential vulnerabilities of these services, and the correlation of these vulnerabilities with the power of the kiosk system configuration.

##### 3.1.1. The Initial Hierarchy

The first step in our work was examining the range of kiosk systems. We initially saw a progressive hierarchy of systems and services:

- *K0 (standalone)*: providing public, seldom-altered information via a standalone kiosk (e.g., a kiosk in a public square that shows the location of stores downtown).
- *K1 (networked)*: providing public, rapidly-changing information via a networked kiosk system (e.g., the above kiosk, modified to also show the latest specials at the stores).
- *K2 (private information)*: providing private, remote information via a networked kiosk system (e.g., the above kiosk, modified to allow customers to make queries about current credit limits and the status of special orders).
- *K3 (transactional)*: allowing clients to change private, remote information via a networked kiosk system (e.g., the above kiosk, modified to allow customers to pay bills electronically by transferring funds from bank accounts to store accounts).

This hierarchy expresses a range of services our customers were considering, and demonstrates several principles:

- that increasing the power of services requires increasing the power of the supporting kiosk system;

- that increasing power increases the potential vulnerabilities and the concomitant security measures (e.g.,  $K_1$  requires protecting networks;  $K_2$  also requires authenticating clients)—the roots of Theorem 1;
- that attacks which increase the power of systems enable attacks that apply to these more powerful systems (e.g., penetrating user interface barriers in  $K_2$  enables data modification attacks natural to  $K_3$ )—the roots of Theorem 2; and
- that an ordering even exists on “power”—the roots of the service ordering.

### 3.1.2. Applying the Methodology

**Inadequacies of the Initial Hierarchy** The structure from Section 2.3.1 easily applies to this hierarchy to produce a simple straight-line graph:

$$K_0 \longrightarrow K_1 \longrightarrow K_2 \longrightarrow K_3$$

Going from  $K_0$  to  $K_1$  adds a network; from  $K_1$  to  $K_2$  adds private information and the need for authentication; from  $K_2$  to  $K_3$  adds the ability to make permanent changes.

However, when we follow the methodology from Section 2.4, we see that this structure is incomplete: the  $\delta$  that characterize the above steps are not quantum, but are independent.

- Whether a kiosk is networked is independent of other service characteristics. Networks add the ability to interact with remote systems and databases—to extend in space the reach of the kiosk. Furthermore, the degree of this extension is not binary, and indeed not even totally ordered.
- Whether a service involves private information is independent of other service characteristics. Furthermore, privacy concerns can be refined to apply to data provided by the client, data returned to the client, or both.
- Whether a service involves permanent changes is independent of other characteristics. Furthermore, defining what constitutes a “permanent change” is tricky. In some sense, all services involve permanent changes, since we cannot roll back the user’s experience. However, if we limit “change” to databases, we may still include simple informational services that are logged, and we may exclude services (such as generating a ticket) that change states in ways not easily described in terms of database writes. (We have not even mentioned the ambiguity introduced by re-using the distributed systems term “transaction.”)

Furthermore, the hierarchy fails entirely to describe proposed services that involve a kiosk session initiating an action whose duration exceeds that of the session.

**Addressing the Inadequacies** Following the methodology, we address these shortfalls by decomposing the differences in the hierarchy into six steps that are independent, and are closer to being quanta. To simplify the presentation, we consider these properties to be binary.

This decomposition yields ■ set of axes that describe the space of electronic services we had been considering. (However, this description is not unique, since spaces can be described by multiple sets of axes.) To keep our presentation tractable, we use high-level, somewhat subjective criteria.

Three of the six steps characterize the type of *activity* in the service:

- **Spatial Extension.** Does the service’s reach extend to machines other than the immediate kiosk?
- **Temporal Extension.** Does the duration of the electronic actions initiated by a service session extend beyond the duration of the service session?
- **Permanence.** Upon successful completion, does the service make permanent (and significant) changes to the system?

The other three steps characterize the type of *data* in the service:

- **Sensitivity.** Is it critical that the system provide correct service?
- **Input Privacy.** Does the user provide private information as part of the service session?
- **Output Privacy.** Does the system provide private information to the user as part of a service session?

This set induces  $2^6$  different service types: a type  $K_P$  for each subset  $P$  of the six properties. Following Section 2.3.1, we order these types using a lattice, putting  $K_P < K_{P'}$  when  $P \subset P'$ .

We offer examples for a few service types:

- **Service Type:**  $\emptyset$

**Example:**  $K_0$ , a standalone kiosk which a user queries for inconsequential public data which is returned immediately.

- **Service Type:**  $\{sensitivity\}$

**Example:** a kiosk like the one above, that instead provides emergency exit locations.

- **Service Type:**  $\{spacial\ extension\}$   
**Example:**  $K_1$ , discussed above.
- **Service Type:**  $\{spacial\ extension, input\ privacy\}$   
**Example:**  $K_2$  above, for special order status, where users authenticate themselves with passwords.
- **Service Type:**  $\{spacial\ extension, input\ privacy, output\ privacy\}$   
**Example:**  $K_2$  above, for credit limit information.
- **Service Type:**  $\{spacial\ extension, permanence, input\ privacy, output\ privacy, sensitivity\}$   
**Example:**  $K_4$  above.
- **Service Type:**  $\{sensitivity, input\ privacy\}$   
**Example:** a standalone kiosk through which a user accesses government services by entering private information, which the kiosk uses to fill out and print an application form (which the user then mails).
- **Service Type:**  $\{spacial\ extension, temporal\ extension, sensitivity, input\ privacy, output\ privacy\}$   
**Example:** a networked kiosk through which a user accesses government service by entering private information, which the kiosk submits electronically. The response is returned via physical mail later.

We provide a more a thorough enumeration of examples in Chapter 1 in [5].

### 3.2. Kiosk Vulnerabilities

In Section 3.1.2 above, we presented  $2^6$  different classes of kiosk service. Enumerating the vulnerabilities associated with these types of services in a brute force or ad hoc way would be exhausting, and would also fail to communicate the tradeoffs between service features and service risks.

However, Section 3.1.2 presented these  $2^6$  classes by organizing them into a partial order (which in this case consists of a simple lattice). We can then follow our methodology by using this structure to clearly and concisely enumerate the vulnerabilities. In this section, we illustrate this by a set of quick examples.

**Basic Vulnerabilities of the Minimal Service** The minimal kiosk service is subject to physical and electronic attacks on the machine itself, in order to deny service or to change the information displayed.

**Basic Vulnerabilities of Intermediate Services** We then enumerate the additional vulnerabilities brought on by moving upward in the lattice. We cite an incomplete list of examples:

- **Spatial Extension.** Networked systems are subject to the vulnerabilities of disruptions, alteration, or eavesdropping of communications.
- **Temporal Extension.** Systems that permit users to initiate services that persist longer than the kiosk session introduce vulnerabilities associated with authentication, rights, and ease of launching these spawned processes. For example, temporal extension may permit using a brief session to launch attacks consisting of computationally-intensive and lengthy processes—attacks which temporal containment would render infeasible by forcing the perpetrator to remain exposed at the kiosk.
- **Permanent Changes.** Systems which permit users to make substantial, permanent changes to system state are directly subject to violation of integrity errors.
- **Input Privacy** Systems that require users to enter private information are subject to the vulnerability of exposure or alteration of that information.
- **Output Privacy.** Similarly, systems that return private information to the user are also subject to the vulnerability of exposure or alteration of that information.
- **Spatial Extension and Privacy.** A networked kiosk system which involves private information risks exposing this information via the network.
- **Temporal Extension and Output Privacy.** A system in which users launch services that later return private information risks exposing this information to unauthorized persons, because user authentication performed for the service session may no longer apply.

**Upward Inheritance** Following Theorem 1, a kiosk system with some set  $P$  of the properties from Section 3.1.2 will be subject to the vulnerabilities from any  $P' \subseteq P$  in the enumeration of basic vulnerabilities..

**Downward Inheritance** Following Theorem 2, a kiosk system with some set  $P$  of the properties from Section 3.1.2 will also be subject to the vulnerabilities from any  $P' \supseteq P$ , providing an enabling attack exists that adds the properties  $DIFF(P', P)$ . For example, a kiosk system consisting of a customized Web browser

may enforce temporal containment through a line of code that terminates all applets when the user completes his or her session. An attacker who has access to the internals of the software might add temporal extension—and thus enable new forms of attack—by disabling this constraint.

### 3.3. Kiosk Points of Attack

Another aspect of vulnerability analysis consists of enumerating the points of attack in a system (and then considering how the vulnerabilities of this system can be realized via threats carried out at these points of attack).

**The Points of Attack** For example, a networked kiosk system might consist of the following components:

- a set of kiosks, each of which consists of a physical shell housing the internal kiosk operating system and software, accessed via the user interface;
- a network connecting the kiosks to each other and to the central host; and
- various remote sites, connected to the central host via remote lines.

Each of these components suggests a point of attack:

- the kiosk user interface;
- the physical environment of the kiosk;
- a fake kiosk;
- the internal kiosk software;
- the network;
- the insider who writes and controls the software at the kiosks and the central site;
- the remote lines.

**An Initial Attempt to Enumerate Threats** The next step is to enumerate the threats possible at each point of attack. However, we found in our practical work that using the above set alone leads to a disorganized, awkwardly referential presentation. For example, exhaustively enumerating the threats that can be carried out at the kiosk user interface includes listing threats such as the following:

- Using ■ bug or trapdoor to gain access to the operating system, then adding a Trojan Horse to tamper with how user data is processed.
- Using a bug or trapdoor to gain access to the operating system, then using this control to penetrate the software front-end at the main host, and then plant Trojan Horses there.

Enumerating the threats applicable at the kiosk software point of attack, the network point of attack, and the insider point of attack requires referring back to this subset of the user interface threats, with implied modifications.

Exhaustive enumeration leads to an extreme lack of clarity. Certain points of attack have natural threats. Not recognizing the inherent structure in the vulnerability set can result in burying these natural threats in the lists compound attacks at other points of attack. In the above example, the natural threats possible at the kiosk operating system level are never explicitly listed. The reader would go to the OS row, only to find a pointer back to the user interface row, where sufficiently close inspection reveals the natural OS attacks buried in compound attacks of the above form.

Not recognizing the inherent structure in the vulnerability sets also leads to a lack of flexibility. Suppose one discovers another attack possible for an adversary with operating system access. This attack makes new compound attacks possible. Since no central list exists of natural OS attacks, one must manually generate and insert all the new compound attacks throughout the list. This disorganization introduces many opportunities for error.

Similarly, suppose one wishes to use the enumeration of vulnerabilities to consider countermeasure. One can defend against ■ compound attack of the above forms by defending against the natural attack at the OS level, and also by defending against the enabling attack that granted access to the OS. Exhaustive enumeration will not make this defense structure clear—the defenses against the natural OS attacks will be buried in defenses against compound attacks, scattered throughout the table.

**Applying the Methodology** Our approach to constructing taxonomies of services and vulnerabilities provides a way to cleanly and concisely enumerate these threats. This applicability is suggested by the very structure of the above-quoted threats: “subvert the difference between the user interface and the kiosk software, then carry out a kiosk software attack.”

To follow our methodology, we construct the service order by enumerating the quantum steps:

- The user interface precedes the kiosk software, since any attack possible via the user interface is possible by accessing the internal software. A bug or trapdoor in the user interface is an attack that subverts the difference.
- The physical kiosk similarly precedes the kiosk software. Physical penetration is an attack that subverts the difference.

- A fake kiosk similarly precedes the kiosk software. Copying software and connecting to a real kiosk (or its network) are attacks that subvert the difference.
- The kiosk software precedes the insider, since anything that can be done via tampered software can be done by an insider. Penetration of the software front-end to provide full system access is a potential attack subverting the difference.
- The network similarly precedes the insider. Penetration of the network front end to provide full system access is a potential attack subverting the difference.
- The remote line similarly precedes the insider. Penetration of the front end on this connection is a potential attack subverting the difference.

We can enumerate the threats possible at these points of attack simply by:

- moving up the order and enumerating the basic attacks possible at each point;
- noting that Theorem 1 means that any attack possible at lower point is possible at a higher point; and
- noting that Theorem 2 means that attacks at higher points (e.g., using software access to plant Trojan Horses) can be carried out at lower points (e.g., the user interface) by first carrying out an attack subverting the difference (e.g., using a bug or trapdoor in the user interface).

In Chapter 2 in [5], we use this new approach to provide a more complete enumeration.

## 4. Conclusions

Using the inherent organization of a service set to enumerate vulnerabilities removes redundancy and awkwardness from an ad hoc or brute force approach.

- Collecting in one spot the natural threats for a point of attack makes it easier to communicate them, and to add and delete threats.
- Using the structure of a system to bring out the compound and inherited structure of many threats makes it easier to produce a complete list, and to construct structured countermeasures.
- Using the inherent structure of threats to organize a listing leads to a more concise presentation. Suppose we have a linear order of  $n$  points of attack, each with  $k$  natural threats and  $r$  subverting

attacks that transform the point of attack to the next level. A brute force enumeration requires listing  $\Omega(kn^2 + kr^n)$  threats.<sup>1</sup> Our methodology lets us specify this complete set with by listing only  $O(nk + nr)$  threats.

As we discussed in Section 1, we offer this work as an attempt to fill a perceived gap. One area of future work will lie in refining and validating our proposed methodology: not against specific services but against classes of services. How well does our session security taxonomy express risks in various approaches to payment over the Internet, or to the latest Web-based front-ends to existing information and data services? Other promising areas may lie in identifying and addressing other gaps.

## Acknowledgments

We are grateful to our colleagues Gary Christoph, Howard Gobioff, Judy Hochberg, Marianna Kantor, Mike Murphy, Tony Warnock, and Bonnie Yantis for their helpful comments and advice.

## References

- [1] S. Adelson, R. Rivenburgh and S.W. Smith. *Enforcing Closure of Subwebs and Expansive Web Sites*. Los Alamos Unclassified Release LA-UR-95-4410, Los Alamos National Laboratory. December 1995.
- [2] D. Bell and L. LaPadula. *Secure Computer Systems: Mathematical Foundations and Model*. Technical Report M74-244, MITRE Corporation. May 1973.
- [3] D.E. Denning. "A Lattice Model of Secure Information Flow." *Communications of the ACM*. 19: 236-243. May 1976.
- [4] D. Fiske. "Another ATM Story." *The Risks Digest*. Volume 6, Issue 66. 21 April 1988.
- [5] J. Hochberg, S.W. Smith, M. Murphy, P. Pedersen, and B. Yantis. *Kiosk Security Handbook*. Los Alamos Unclassified Release LA-UR-95-1657, Los Alamos National Laboratory. May 1995.
- [6] G. Morris, T. Sanders, A. Gilman, S. Adelson, and S.W. Smith. *Kiosks: A Technological Overview*. Los Alamos Unclassified Release LA-UR-95-1672, Los Alamos National Laboratory. May 1995.
- [7] S. W. Smith. *Secure Distributed Time for Secure Distributed Protocols*. Ph.D. thesis. Computer Science Technical Report CMU-CS-94-177, Carnegie Mellon University. September 1994.
- [8] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

<sup>1</sup>The  $i$ th level has  $k$  natural threats,  $k(i - 1)$  inherited threats, and  $k \sum_{j=1}^{n-i} r^j$  subversion threats. The  $\sum_i k(i - 1)$  gives the  $kn^2$ , and the  $\sum_i \sum_j r^j$  reduces to  $\sum_{l=1}^n lr^{n-l}$  which we can bound below by  $r^n$ .



# WWW Electronic Commerce and Java Trojan Horses

J. D. Tygar                          Alma Whitten  
`tygar@cs.cmu.edu`    `alma@cs.cmu.edu`  
*Carnegie Mellon University*  
*Pittsburgh, PA 15213*

## Abstract

*World Wide Web electronic commerce applications often require consumers to enter private information (such as credit card numbers) into forms in the browser window. If third parties can insert trojan horse applications onto a consumer's machine, they can monitor keyboard strokes and steal private information.*

*This paper outlines a simple way to accomplish this using Java or similar remote execution facilities. We implemented a simple version of this attack. We give a general method, window personalization, that can thwart or prevent this attack.*

## 1 Introduction

Computer security experts have long recognized the threat of trojan horse programs [6, 15, 13]: programs that appear to perform one function while actually performing a second, unwanted function. A particular concern arises from the presentation of an interface to a user or consumer. Since human users identify applications by their interface, a human user may be unable to distinguish a legitimate program from a rogue program's mimicking of the first program's interface. These concerns are further exacerbated when the consumer is using electronic commerce protocols on the World Wide Web. The consumer may be required to enter security crucial information (such as credit card numbers, bank routing and checking account information, billing account information, personal demographic data, etc) into the local client. If a trojan horse can grab this information or fool the consumer into submitting

this information to a third party rather than to a valid electronic commerce server, then the consumer can unintentionally release confidential information to third parties. This paper gives an example of how remote execution systems such as Java can easily host such a trojan horse attack. We then give a general method of *window personalization* to address this problem.

### 1.1 Trojan horses

Consumers face two types of trojan horse challenges when they engage in electronic commerce on the World Wide Web.

#### 1.1.1 Bogus remote pages

A consumer may be looking at a forms page other than the one she believes she is communicating with. For example, although the consumer may believe that she is communicating with Firm X, she may in fact be communicating with another party. URLs are not easy to check, and there have been a number of instances of prank URLs parodying or imitating WWW sites. (For example, <http://www.dole96.com> is Bob Dole's official 1996 presidential campaign site. On the other hand, <http://www.dole96.org> is a humorous parody of the Dole election. Reportedly, many users have been fooled.) In the credit card industry, this is a serious concern — today, fraudulent use of credit cards *by merchants* is already recognized to be a serious problem (this is sometimes called the "Lynndon LaRouche problem" after the fringe presidential candidate who has been accused of credit card fraud.) To address this problem, Visa and Mastercard, in their Secure Electronic Transaction (SET) specification [11], indicated that merchants (both legitimate and bogus) should not receive credit card numbers and similar confidential billing information about consumers. Rather, credit card information entered by consumers should be encrypted before

---

This work was supported in part by Defense Advanced Research Project Agency (ARPA contract F33615-93-1-1330), the National Science Foundation (NSF cooperative agreement IR-9411299), the US Postal Service, and Visa International. This work is the opinion of the authors and does not necessarily represent the view of their employers, funding sponsors, or the US Government.

it leaves the local client, so that only an acquiring bank could read the information. Thus, SET addresses the bogus page attack, but leaves open the possibility of local trojan horses.

### 1.1.2 Local trojan horses

A consumer may have a local trojan horse running on her machine. This type of local trojan horse can:

- imitate remote pages, or
- grab keystrokes (such as SET entry keystrokes) from the local machine. (The possibility of this type of attack has been known in the computer science folklore for many years; recently Nathaniel Borenstein gave a dramatic demonstration of trojan horse keystroke grabbing [2].)<sup>1</sup>

In the past, consumers have been protected by the relative difficulty of loading trojan horse applications onto arbitrary workstations. If the consumer took moderate care to protect herself from loading untrusted software (including viruses and worms), she could reduce the risk of attack to a manageable level.

However, local trojan horses are particularly dangerous for electronic commerce protocols that depend on the local client interfaces to obtain and securely handle confidential information. Examples of systems that use these features include SET (discussed above) and NetBill[16, 3] For example, if a trojan horse emulation of the SET interface is very well done, it may be difficult or impossible for the consumer to determine whether she is dealing with the true program or not. Such a rogue program could transmit the consumer's credit card number directly to the adversary; alternatively, it could store it for later retrieval.<sup>2</sup>

<sup>1</sup>In his First Virtual work [14, 8], Borenstein and his colleagues argue for a system where a consumer would enter a First Virtual account number, rather than directly entering credit card information. Consumers would have the opportunity to review and decline charges that were improperly made on their account. While this certainly protects the credit card account, there is still an issue of providing maximum protection of the First Virtual account number from trojan horse attack.

<sup>2</sup>Strictly speaking, this type of attack is not limited to electronic commerce protocols. Indeed, there appears to be no limit to the possible mischief that can be caused by trojan horse programs that infiltrate a consumer's machine. A trojan horse might emulate a word processor, causing (potentially sensitive) text to be transmitted to an adversary. It might infect an e-mail system allowing messages to be read by third parties. It might infect a compiler allowing trojan horses to be inserted in locally compiled programs [17]. Ultimately, any application program might be emulated. However, the case of

Java applets[10] have changed the balance of power — increasing the possibility of untrusted software and local trojan horses being transferred to a workstation. Java applets are usually loaded onto a workstation without requiring explicit applet-by-applet consent by the consumer. If a Java applet created by an adversary is loaded onto the consumer's workstation, the applet can put arbitrary images on the screen within the browser window and can communicate keyboard information back to a host. A consumer may not be able to distinguish the trojan horse applet display from a valid information request. Thus, Java applets can easily serve as trojan horses that mimic electronic commerce applications. Java is not the only culprit. Other *remote execution* mechanisms such as Omniware [1], Telescript [12], and Dyad [18] provide ample opportunities to download programs to a consumer's computer that can display arbitrary graphical interfaces and transmit information to an adversary. In section 3 below, we discuss an example trojan horse Java applet that we developed which performs its own emulation of a standard Netscape dialogue box. Normally, consumers expect Java applets to display windows and dialogue boxes only with a warning bar underneath them. By writing our own graphics routines, we were able to make our applet bypass the standard Java window library and write directly to the screen. This makes our Java applet output bit-for-bit indistinguishable from the standard Netscape I/O display boxes. As we discuss in section 2 below, standard Java security<sup>3</sup> mechanisms are useless for protection against this type of attack.

## 1.2 Paper outline

This paper outlines a simple way to accomplish a local trojan horse attack using Java or similar remote execution facilities. We show an implementation in Java of a simple example of this type of trojan horse.

We propose a new mechanism of *window personalization* that can thwart this type of attack. Window personalization allows a consumer to select a pattern for window display that will be unknowable (or very difficult to determine) by rogue applets and other

electronic commerce is particularly compelling since there is a well-defined target of attack — the consumer's credit card numbers or bank account information.

<sup>3</sup>Recently, Dean, Felton, and Wallach [5] and others have argued that substantial flaws exist in the Java API library and in the methods used to check type safety. These flaws lead to direct attacks that can be implemented in Java applets. Our work is orthogonal to these flaws: even a hypothetical "perfectly secure" version of Java would permit these types of attacks. We discuss this further in section 2.

transmission media for trojan horses. Through window personalization, a consumer can catch almost all trojan horse emulations of human interfaces; the rogue program can not determine the correct way to format the interface, and so it is likely to appear wrong to the consumer. In section 5 we show how this method can be extended to other applications such as point-of-sale transactions and automatic teller machine transactions.

## 2 Java

Java is a programming language and associated set of development tools developed by Sun Microsystems. Similar to C++ in syntax, Java is intended for writing programs that will be runnable on a variety of dissimilar computer architectures. This is accomplished by compiling Java programs into an machine-independent byte code format, which can then be run on any computer equipped with a Java interpreter.

Java can be used to write stand-alone applications, but it has gained its widespread popularity by its use to create applets, which are executable objects written in Java and embedded in World Wide Web pages. When a web page containing a Java applet is loaded into a Java-enabled browser, the applet's byte code is also loaded, interpreted and executed within the browser. Applets can be used to create web pages containing animation loops, arcade games, and other highly interactive applications which would have been infeasible if executed from the server due to latency and server load.

Executing applets in a secure manner must necessarily be the responsibility of the client's web browser. Java-enabled web browsers are expected to make use of Java's well defined type system in order to verify that the byte code they receive corresponds to a valid Java program. The Java language also includes an extensive application program interface (API) specification, which is the only way that applets can perform any input/output or other system functions. The web browser must include an implementation of the Java API class libraries, with appropriate restrictions on what applets, as untrusted code, are allowed to do.

In this discussion, we will consider Netscape Navigator as our example of such a Java-enabled web browser. Navigator incorporates three significant security restrictions in its version of the Java API:

1. No access to the local filesystem is allowed.
2. Socket connections are allowed only to the

server on which the web page containing the applet resides.

3. Windows and dialogue boxes opened by applets contain vivid banners labeling them as "Untrusted Java Applet Window".

Recent work by Dean, Felten and Wallach [5] has focused on security flaws in Netscape Navigator Java which are due to implementation errors, such as significant weakness in the Java class loader. These flaws may be exploited by a sophisticated attacker in order to bypass any or all of the above API security restrictions. In our work, by contrast, we have chosen to assume a correct implementation of Java in which the intended security restrictions are perfectly enforced, and to show that there exists a particular type of attack on security which is not prevented or even significantly hindered by these restrictions.

We begin by pointing out that Java applets are allowed complete control over the screen pixels that fall within their allotted area of the browser window, and that it is unrealistic to expect that any significant limits could be placed on this control while still allowing applets to perform animation. Similarly, applets have full access to information about mouse and keyboard events that occur within that area, and that access cannot be limited much without severely hampering the interactivity that applets are intended to provide. The combination of these two abilities allows applets to bypass the third security restriction with relative ease, since it is a simple matter for an applet to draw and manage its own dialogue box within the confines of the browser window. Not only would such a dialogue box have no warning banner, it could have any appearance that the applet's designers desire.

If a rogue applet is successful in stealing information from its host environment, whether by tricking it out of the consumer or through some other method, the second security restriction does little to prevent the applet from transmitting its stolen information over the network. The applet may, with perfect legitimacy, connect back to its server via socket and report what it has stolen; alternatively it may simply make an http request to any machine on the network and include its stolen information as part of the request string.

In summary, even given a hypothetical perfect implementation of Java and the listed Navigator security restrictions, a rogue applet is still free to trick information out of the consumer through clever control of its area of the screen display, and can easily communicate such stolen information over the network to any accomplice it desires.

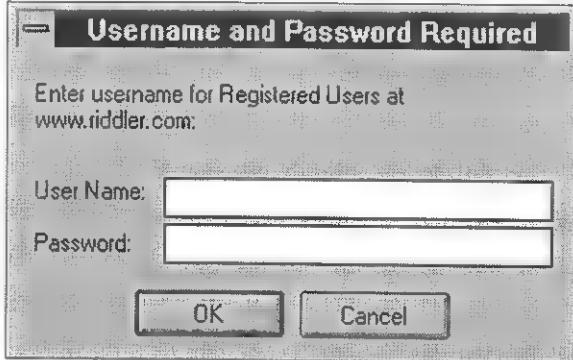


Figure 1: Authentication dialogue box displayed by WindowsNT Netscape browser

## 2.1 Signed Applets

A great deal of attention has recently been paid to the strategy of increasing Java security by having applets certified and digitally signed by some trusted authority. Current proposals for this argue for certification only as a method for establishing the identity of the code provider; it is expected but in no way guaranteed that code providers with reputations to protect will take appropriate care to verify the safety of the code they sign. Given that Microsoft, a company that has announced interest in Java code signing, has in the past released software containing hostile code [7], we believe that such certification is best used in conjunction with additional security strategies, such as the window personalization technique described in this paper, which is complementary to (and independent of) applet signing.

It may also be the case that we will see the further development of the code signing strategy beyond simple certification of provider identity to encompass at least limited verification of code safety and other attributes. In the event that this occurs, we once again note that even the most competent and conscientious certification authority is likely to occasionally err and certify a malicious applet as safe. Given the extremely sensitive nature of some of the information that such applets might attempt to steal, it clearly remains advisable to minimize that risk through additional strategies where possible.

Finally, we point out that there will be substantial pressure on many web users to allow some uncertified applets to run; the broad variety and large number of applets present on the Web today argues that applets are fulfilling a need — and requiring and checking digitally signed certification for every applet is likely to be viewed as being a logically complex problem.

## 3 Example Attack

We here describe a very simple example of an attack using a trojan horse Java applet. The reader with imagination will have no trouble constructing a more elaborate example that would be sensitive to the specific browser and client platform used, and could do a broader range of sophisticated attacks.

The Netscape Navigator browser uses a standardized dialogue box to request username/password pairs when a web server responds to an http request with a demand for authentication. Figure 1 shows the appearance of the authentication dialogue box used by WindowsNT Netscape. To demonstrate the use of Java for trojan horse attacks, we have written an applet which fakes the appearance and behavior of this dialogue box; for the interested reader, this applet can be found at <http://blind.trust.cs.cmu.edu/spoof.html>.

The applet attempts to trick users into believing that they are authenticating themselves as usual to a particular web site. This is done by mimicking the appearance and behavior of the real site within the browser window. If the user clicks on a link that would cause the real site to request authentication, the applet displays its fake authentication dialogue box; clicks on other links are handled by loading the correct pages from the real web site. If the fake dialogue box fools the user into entering their username and password, the applet sends the stolen information back to a waiting process on `blind.trust.cs.cmu.edu` via socket connection, tells the user that the web site server is not responding, and loads the real web page from the real web site, so that the user's next attempt will succeed.

Implementing this applet required fewer than 200 lines of Java code<sup>4</sup>. As shown in Figure 2, our ap-

<sup>4</sup>Java source code for our applet is available at <http://blind.trust.cs.cmu.edu/spoof.class>.

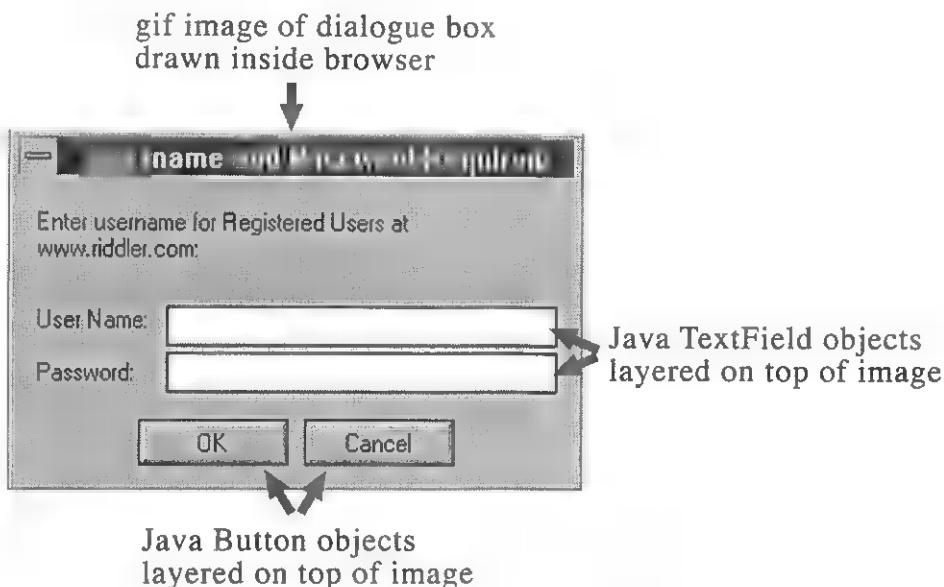


Figure 2: Our trojan horse version of the authentication dialogue box

plet uses a simple image of an actual authentication dialogue box, acquired by screen capture and edited in Adobe Photoshop, to mimic the appearance of the dialogue box within the browser window. Actual Java **TextField** objects are drawn on top of the image to provide the necessary pair of editable text fields. We overloaded the applet event handling routines in order to make the simulated buttons on the dialogue box behave appropriately in response to mouse events, including performing the necessary socket communication and http request when either button is actually pressed.

This applet was created in order to demonstrate a point, and as such, it does not currently present a seamless attack. For example, the alert user may notice that the dialogue box is not druggable, or that the Location URL displayed by Netscape is not what the user expected (however, this is located in a position where many users might ignore the information displayed by Netscape). Some of the current limitations could be fixed by putting more work into the applet, while others, like the Location URL, are effectively enforced by Netscape.

This attack can easily be generalized to mimic any user interface element that has a standardized appearance and requests sensitive information from the user; some other obvious examples of such information are credit card numbers for electronic commerce applications, account numbers for banking, or, in the case of hospitals, a wide variety of potentially sensitive medical information.

## 4 Window Personalization

Trojan horse attacks such as we have described rely on imitating the visual appearance of some program that the consumer already trusts with sensitive information. Standardized user interface appearance is a great help to the designers of such attacks, since it removes the problem of determining what appearance the consumer expects.

We propose that programs which require the consumer's trust should employ window appearances which are easily recognizable to the consumer yet difficult to predict by an attacker. The obvious mechanism for accomplishing this is to require the consumer to personalize the appearance of the software at the time the trust relationship is formed, which may be installation or account initialization depending on the application involved. Figure 3 shows a hypothetical user's personalization of the Netscape dialogue box with a Batman logo. Our trojan horse attack would not be able to predict this personalization, and its ability to fake the appearance of the real dialogue box would be substantially weakened.

As another example, consider a SET dialogue box. If the dialogue box has a standardized, predictable appearance, it will be an easy target for a similar trojan horse interface-emulator applet. On the other hand, suppose that the consumer is able to select a background display for the dialogue box (out of a range of thousands of potential displays). As long as the Java applet is not able to determine what

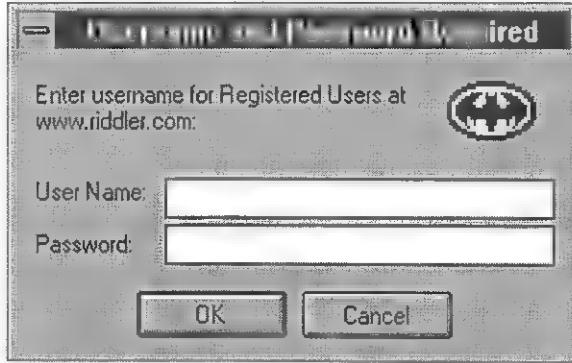


Figure 3: Authentication dialogue box protected by personalization

the background display expected by that particular consumer is, the applet will not be able to credibly simulate the user interface of the program under attack.

In order for this strategy to be most effective, the trusted application should not only require the consumer to participate in a personalization process, but should, within that process, strongly convey to the consumer the importance of not trusting any window that does not display characteristics in conformance with the personalization. The process should also be designed to maximize the unpredictability of the consumer's personalization choices; for example, when offering the consumer a choice of window backgrounds, it would be best to offer more than just two or three backgrounds to choose from, and to present the possible backgrounds in a randomized ordering or layout to compensate for the fact that most consumers may simply choose the first option they are given.

Window personalization is most effective against attackers who either do not have an opportunity to find out the potential victim's personalization choices, or who are not interested in creating a trojan horse designed to fool a specific individual.

## 5 Extensions

In the more general case, window personalization may be applicable to any situation in which users need to confirm that an interface is being presented by an entity with which they have a prior trust relationship, and not by an imposter.

As one example, there have been a number of publicized cases of fake automatic teller machines (ATMs), set up in public places and used to steal card and PIN information from unsuspecting customers. If window personalization were employed,

the ATM could be expected to display the user's personalized window style at the time it requests the entry of the PIN number.<sup>5</sup> The absence of the correct window style would be a signal that the machine did not have access to the bank database of window styles corresponding to cards, and therefore should be regarded with suspicion.

A similar problem arises in the case of point-of-sale (POS) transactions, in which the user must communicate with a trusted entity, the bank or perhaps the stored value card, via untrusted POS equipment belonging to the merchant. A corrupt merchant might have modified the POS equipment to display a false charge amount, in an attempt to trick the customer into entering a confirmation for a charge which is actually larger than that which appears on the POS display. If, however, a personalized display style is a shared secret between the user and the trusted entity, then the user can take the display of the charge amount in the correct style as confirmation that the amount displayed is the actual charge according to the trusted entity, even though it is displayed on untrusted equipment.

The careful reader will note that these examples are not fully satisfactory. He will wonder: why can't a trojan horse POS system perform a "man in the middle" attack — actually connecting to the POS network, observing the message transmitted in both directions, and recording the information for later pickup? That attack is certainly possible, although it requires substantial preparation; the attacker would need to connect to the POS network

<sup>5</sup>Note: This differs substantially from the way that ATM machines operate. Typical ATM authentication today relies only on local encrypted information stored on a single ATM card. No central database query is done until after authentication is complete [4]. The solution discussed here would require an initial lookup by a centralized machine of the user's particular window personalization style, making the ATM protocol less efficient.

(or subvert an existing POS), properly authenticate the bogus POS machine to the bank, and then interpret formatting messages coming from the bank and properly adjust the display presented to the user. While this might perhaps be possible, it certainly would require a much higher level of skill to successfully pull off this attack.<sup>6</sup>

## 6 Conclusions

The popularity of Java applets means that a vastly increased number of users will run untrusted software on a regular basis. Current security strategies that focus on limiting applet access to dangerous system calls are useful but insufficient; the ability of applets to freely manipulate even a portion of the screen display, combined with the ability to send information back to their source machines via the network, allows applets to mount trojan horse attacks by imitating the user interface elements of trusted software. It is infeasible to increase security by further limiting those applet abilities, because doing so would greatly limit the ability of applets to provide the interactivity and animation on which so much of their appeal is based.

Window personalization is a supplementary security strategy which is independent both of the strategies described above and of code signing. If users are strongly encouraged to personalize the appearance of the user interface elements of their trusted software, in ways that are highly recognizable to the user yet very difficult to predict by others, then the designers of rogue applets will not be able to mimic those user interface elements convincingly because the personalized aspects of the appearance will be unknown to them.

The window personalization strategy can be extended to any situation in which a user needs the ability to verify that a user interface is being presented by a trusted entity and not by an imposter. Two examples in which this need for verification may arise are automatic teller machines and point of sale transactions.

## References

- [1] A. Adl-Tabatabai, G. Langdale, S. Lucco, and R. Wahbe. Efficient and language-independent mobile programs. In *Proceedings of the 1996 ACM SIGPLAN Symposium on Programming*

<sup>6</sup>Even smartcards can not completely solve these problems; for a discussion of the difficulties with the smartcard approach to solving the POS integrity problem, see [9].

*Language Design and Implementation*. ACM Press, May 1996.

- [2] Nathaniel Borenstein. Vulnerability of software based credit card encryption. At <http://fv.com/ccdanger/index.html>; see also *San Jose Mercury News*, 29 January 1996, "Program shows ease of stealing credit information" by Simpson L. Garfinkel.
- [3] Benjamin Cox, J. D. Tygar, and Marvin Sirbu. NetBill security and transaction protocol. In *Proceedings of the First USENIX Workshop in Electronic Commerce*, pages 77–88, July 1995.
- [4] Donald Watts Davies and W. L. Price. *Security for Computer Networks: an Introduction to Data Security in Teleprocessing and Electronic Funds Transfer*, 2nd Edition. Wiley, 1989.
- [5] Drew Dean, Edward W. Felten, and Dan S. Wallach. Java security: from HotJava to Netscape and beyond. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 190–200, May 1996.
- [6] Dorothy Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.
- [7] Peter J. Denning. *Computers Under Attack: Intruders, Worms, and Viruses*. ACM Press, New York, N.Y., 1990.
- [8] N. Borenstein *et al.* Perils and pitfalls of practical cybercommerce: The lessons of First Virtual's first year. In *Proceedings of Frontiers in Electronic Commerce*, October 1995.
- [9] Howard Gobioff, Sean Smith, J. D. Tygar, and Bennet Yee. Smartcards in hostile environments. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, November 1996.
- [10] James Gosling and Henry McGilton. The Java language environment: A white paper. Technical report, Sun Microsystems, May 1996. See also [http://www.javasoft.com/doc/language\\_environment/](http://www.javasoft.com/doc/language_environment/).
- [11] Mastercard International and Visa International. *Secure Electronic Transaction (SET) Specification*, June 1996. See <http://www.visa.com> or <http://www.mastercard.com>.
- [12] General Magic. Telescript technology: The foundation for the electronic marketplace.

Technical report, General Magic, 1996. See also  
[http://www.genmagic.com/Telescript/  
Whitepapers/wp1/whitepaper-1.html](http://www.genmagic.com/Telescript/Whitepapers/wp1/whitepaper-1.html).

- [13] Peter G. Neumann. *Computer Related Risks*. ACM Press and Addison-Wesley, 1995. Also see Risks Digests at <ftp://ftp.sri.com/risks>.
- [14] Darren New. Internet information commerce: The First Virtual approach. In *Proceedings of the First USENIX Workshop in Electronic Commerce*, pages 33–68, July 1995.
- [15] Adrian R. D. Norman. *Computer Insecurity*. Chapman and Hall, London, 1983.
- [16] Marvin Sirbu and J. D. Tygar. NetBill: an internet commerce system optimized for network delivered services. *IEEE Personal Communications*, pages 34–39, August 1995.
- [17] Ken Thompson. Reflections on trusting trust. In Robert L. Ashenhurst and Susan Graham, editors, *ACM Turing Award Lectures, The First Twenty Years, 1966 – 1985*, pages 163–170. Addison-Wesley, 1987.
- [18] Bennet Yee and J. D. Tygar. Secure coprocessors in electronic commerce applications. In *Proceedings of the First USENIX Workshop on Electronic Commerce*, pages 155–170, July 1995.

# On Shopping *Incognito*

Ralf Hauser\*

McKinsey Consulting  
Zürich, Switzerland  
hauser@acm.org

Gene Tsudik†

University of Southern California  
Information Sciences Institute  
Marina Del Rey, CA  
gts@isi.edu

## Abstract

The increasing popularity and importance of electronic commerce makes very apparent the issue of privacy and anonymity for electronic consumers. Technologies ■ already in existence that offer some degree of electronic privacy, e.g., electronic/digital cash. However, the mere fact that electronic commerce is conducted over existing network infrastructure (such ■ the global Internet) runs counter to the privacy of the consumer. This is due mainly to the end-to-end nature of application protocols that are used as vehicles for electronic commerce – World-Wide Web (WWW), Electronic Mail, File Transfer Protocols and others.

In this paper we discuss the issue of consumer anonymity and propose some solutions for the activities that typically “surround” electronic payment: pre-purchase browsing and merchandise delivery.

## 1 Introduction

The state of the Internet today demonstrates the importance and the proliferation of electronic commerce. In this type of an environment – just as in the traditional “non-electronic” commerce – consumer privacy is becoming ■ major concern. The mere fact that electronic commerce is conducted over an existing open network infrastructure such as the Internet runs counter to the privacy of the consumer. Often, there are legitimate reasons for a party to remain anonymous at least during an initial stage of developing a business relationship with another party. But it becomes very difficult or even impossible for a party to safeguard its identity when addressing another party over an open network. Of course, this issue becomes even more complex when both communicating parties want to stay anonymous.

The problem is essentially due to the end-to-end nature of communication protocols used as a vehicle for electronic commerce. From the outset, the nature of current network protocols and applications runs counter to privacy. The vast majority have one thing in common: they faithfully communicate end-point identification information. “End-point” in this context can denote a user (with a unique ID), a network address, or an organization name. For example, electronic mail routinely communicates sender’s address in the header. File transfer (e.g., FTP), remote login (e.g., TELNET), and hypertext browsers (e.g., WWW) expose addresses, host names and IDs of their users.

\*This work was performed in part while both authors were at the IBM Zürich Research Laboratory, Rüschlikon, Switzerland

†This work was supported in part by the Defense Advanced Research Project Agency (DARPA), Computer Systems Technology Office, under contract DASG60-95-1-002.

Given the inadequacies of current communication protocols, our main goal is to provide ■ solution enabling anonymous data exchange, i.e., a method that would allow a prospective consumer to obtain bids or offers from prospective merchants without disclosing his identity. A closely related scenario is that of a consumer who, having already paid, takes delivery of the (electronic) merchandise while still remaining anonymous with respect to the merchant. Such methods must be resistant to false representations from both sides: the consumer’s as well as the merchant’s. Another goal is to exploit existing services already available in open networks and thus minimize deployment costs.

It is NOT the purpose of this paper to obtain a method for anonymous electronic payment. This topic has been addressed elsewhere, e.g., [3, 5]. More concretely, the two aspects of electronic commerce that are of particular interest:

- Pre-purchase Browsing (PPB) and
- Electronic Merchandise Delivery (EMD)

PPB refers to the electronic equivalent of *browsing*<sup>1</sup> or *window-shopping* in real life. During this stage a consumer typically compares various merchandise and advertised prices; at times a consumer even gets a written and signed estimate (or cost-breakdown) for services or goods. Following the analogy with physical commerce, a prospective consumer should not have to reveal his identity in order to partake in PPB.

EMD is the ultimate stage of an electronic commerce transaction. It involves the actual delivery (communication) of the electronic goods or services such as digital video, audio or text. Once again, provided that the interaction preceding EMD preserves the anonymity of the consumer, there is no reason for it to be sacrificed during EMD.

## 2 Required Infrastructure

Both PPB and EMD are not self-contained, independent mechanisms; they require certain basic security services for proper operation. We identify two main requirements:

1. Public-key certification infrastructure
2. Anonymous message-based communication

They are briefly described below.

<sup>1</sup>Not to be confused with WWW browsing.

## 2.1 Public Key Certification

Public key certification (PKC) serves a multitude of purposes. Currently, its most prominent use is in the area of secure electronic mail. Two of the best known examples are Privacy-Enhanced Mail (PEM) and Pretty Good Privacy (PGP) [7, 10]. The latter and its derivatives is, without a doubt, the most widely used. PEM includes a PKC hierarchy and functions. PGP's certification is somewhat *ad hoc*, although it does allow for informal certification hierarchies.

In the discussions below it is assumed that all relevant parties: merchants and consumers, are equipped with individual public key certificates. At the very minimum, a certificate is assumed to contain the party's name, public key, validity time, and the name of issuing authority. (The issuing authority is often referred to as the "Certification Authority" or CA.)

## 2.2 General-Purpose Anonymous Communication

Anonymous communication is necessary in order to preserve anonymity of the sender, i.e., the consumer in this context. It can also be used to hide the location of the sender. We note that sender's identity and location are often (but not always) tightly coupled.

Anonymous shopping requires some form of anonymous message-based communication. This communication can be synchronous (real-time, e.g., via WWW) or asynchronous (store-and-forward, e.g., via email.) In case of synchronous communication we assume the existence of a protocol-specific anonymizer or a mix [15]. An anonymizer is essentially a program that functions as a conduit between the real sender and receiver and hides the identity of the sender, e.g., by using a random alias. In case of asynchronous communication, we assume the existence of an anonymizing relay service. One example is the anonymous remailer (AR) – an anonymizing relay for electronic mail. There are at least a dozen Internet ARs offering varying degrees of anonymity and inter-operability [11, 6, 13].

Anonymizers for synchronous communication are not yet widely used<sup>2</sup>. The only currently available tool is the WWW Anonymizer [19] which provides (weakly) anonymous Web access. (It is a single-site service which implies a central point of trust as well as a central point of failure. This is far less secure than the Chaum's Mix approach used by some of the current anonymous remailers.)

Anonymous shopping should place no restrictions on the type of the anonymizing tools. The only exception is the requirement for two-way communication: it should be possible for a consumer to send an anonymous request to the merchant and for the merchant to reply to the consumer. This can be done in a number of ways. One method used by simple (existing) ARs is to set-up a table that maps real email addresses into aliases; this way, a merchant can reply to the alias that is subsequently translated into a real email address by the AR.<sup>3</sup>

<sup>2</sup>It appears that anonymizing synchronous communication has been largely the province of military communications. However, solutions for anonymizing real-time traffic in certain restricted environments (e.g., Ethernet, ISDN) have been proposed [9].

<sup>3</sup>This is, in fact, the exact *modus operandi* of Penet – the first

There are certainly more secure and sophisticated methods such as having the sender (consumer) pre-compute a secret return path which is then "blindly" used by the receiver/merchant. Suffice it to say that, as long as it is available, the exact nature of the two-way anonymous communication is not germane to the discussion at hand.

## 2.3 Notation

The table below illustrates the notation used throughout this paper.

C,M	Protocol participants; consumer and merchant
H()	Strong one-way hash function, e.g., SHA or MD5.
CA	Certification Authority
$PK_x$	Public key of $X$
$SK_x$	Private (secret) key of $X$
$K(text)$	Encryption of "text" under key $K$
$Cert_X$	Public key certificate of $X$ ; includes $PK_x$
$S_x[text]$	Signature computed with $SK_x$ , $S_x[text] = SK_x[H(text)]$
{text}	Optional text
$R_c$	Random number (nonce) generated by party $X$

## 3 Anonymous Pre-Purchase Browsing

### 3.1 Consumer Request

A prospective consumer (C) who wishes to obtain a bid/offer from a merchant (M) for certain merchandise composes the following message:

OFFER\_REQ =DESCR, $SIG_c$

where:

- DESCRIPTOR is the textual description of the desired merchandise including, e.g., quantity, color/size, delivery dates, etc.<sup>4</sup>
- $SIG_c = S_c[DESCR, R_c]$

In other words,  $SIG_c$  is the consumer's signature computed over the hash digest of DESCRIPTOR together with a randomly-generated, used-only-once quantity  $R_c$ .

A crucial detail is that, even though  $R_c$  is used in computation of  $SIG_c$ ,  $R_c$  is NOT included as part of OFFER\_REQ message.

After composing OFFER\_REQ the consumer uses the anonymous communication service to send the message to the merchant.

### 3.2 Merchant Reply

Upon receiving OFFER\_REQ the merchant is not required to perform any security-related activity. Instead, the merchant examines DESCRIPTOR and determines whether or not he can make a corresponding offer or bid. This process depends both on the merchant and on the type of merchandise specified in the incoming request. If and when the merchant is ready to make an offer, he composes the following message:

Internet remailer service located in Finland.[13]

<sup>4</sup>We assume that the consumer already obtained DESCRIPTOR by either contacting the merchant beforehand (perhaps, via an anonymizer) or pulling it out of a catalog.

**OFFER.REP** =  $\{SIG_c\}, \{Cert_m\}, OFFER\_DESCR, SIG_{offer}$   
 where:

- $SIG_c$  is the same as in OFFER.REQ message
- $Cert_m$  is the public key certificate of M
- OFFER.DESCRI is the textual description of the offer including, e.g., current date/time, price, currency type, accepted payment type, delivery schedule, etc.
- $SIG_{offer} = S_m[SIG_c, DESCRI, OFFER\_DESCR]$

The merchant's public key certificate –  $Cert_m$  – is optional in OFFER.REP since it is possible that the consumer already has it in his possession.

The hash function digest of DESCRI and  $SIG_c$  is also optional since its only purpose is to help the consumer match the outstanding OFFER.REQ with the incoming OFFER.REP. This is only necessary in the asynchronous communication model.

### 3.3 Subsequent Actions by Consumer

When OFFER.REP is received, the consumer performs the following actions:

1. Using  $SIG_c$  identifies the outstanding OFFER.REQ.
2. If applicable, verifies the validity, authenticity and data integrity of the merchant's certificate –  $Cert_m$ . (Techniques for doing this are well-known.)
3. Examines OFFER.DESCRI for consistency with the corresponding DESCRI and determines whether to accept the offer.
4. If the offer is not accepted, no further actions are taken.
5. If the offer is of interest, the consumer computes:  

$$PK_m(SIG_{offer})$$
 where  $PK_m$  is the merchant's public key extracted from  $Cert_m$ .  
 and  

$$H(SIG_c, DESCRI, OFFER\_DESCR)$$
6. If the two values match the consumer is assured that the offer is genuine.

Given that the offer is acceptable and genuine, the consumer – if he so wishes – can approach the merchant directly, i.e., without going through the anonymizing process. (The consumer may decide to act upon the merchant's offer some time after the initial exchange, e.g., the next day.) Of course, this is not a requirement, i.e., if the merchandise is digital it can be delivered on-line. In that case, it may be possible and desirable for the consumer to remain anonymous.<sup>5</sup>

<sup>5</sup> Of course, in some cases the consumer will resort to conventional methods of payment (e.g., a credit card) thus sacrificing his anonymity.

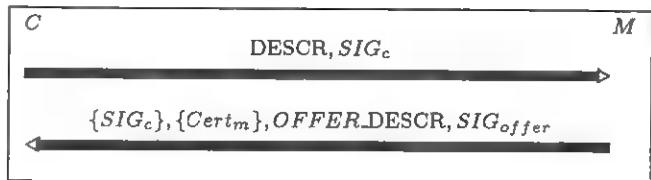


Figure 1: Anonymous Pre-Purchase Browsing Protocol

### 3.4 Subsequent Actions by Merchant

If and when the consumer decides to accept the offer, the merchant needs to establish that the offer is both genuine and still valid. Regardless of how the consumer goes about it, the following information must be communicated to the merchant:

- DESCRI
- $SIG_c$
- OFFER.DESCRI
- $SIG_{offer}$
- $Cert_c$
- $R_c$

The first four are taken directly from the aforementioned OFFER.REQ and OFFER.REP messages.  $Cert_c$  is the public key certificate of the consumer and  $R_c$  is the hereto secret quantity that was used to compute  $SIG_c$  (see above.)

In order to establish the validity of the bid, the merchant does the following:

1. Verifies the validity, authenticity and data integrity of the consumer's certificate –  $Cert_c$ .
2. Extracts  $PK_c$  from  $Cert_c$ .
3. Computes  $PK_c(SIG_c)$  and  $H(DESCR, R_c)$
4. If the two values match the merchant is satisfied that  $SIG_c$  is the genuine signature of the consumer C.
5. Finally, M computes:  

$$S_m[H(SIG_c, DESCRI, OFFER\_DESCR)]$$
6. If this value matches  $SIG_{offer}$ , M is satisfied that the signature (and, hence, the corresponding offer) is genuine.

### 3.5 Maintaining Consumer Privacy

In the event that the consumer objects to revealing his identity to the merchant (even after deciding to go ahead with the payment), a mutually-trusted third party can be employed to verify the validity of the offer instead of the merchant. An obvious choice of such a third party is an *Acquiring Gateway* or simply *Acquirer* – ■ entity

that in any case gets involved in the payment process. In this case,  $R_c$  and  $Cert_c$  must be communicated to the third party bypassing the merchant (either physically or by means of encryption.) This is almost exactly the procedure in the *iKP* electronic payment protocol [3, 4].<sup>6</sup>

A much simpler way of maintaining consumer anonymity is to forgo consumer certification (at least in the context of the current discussion.) Then, the protocol has to be revised to replace  $SIG_c$  with  $H_c$  (see Figure 2) where  $H_c = H(\text{DESCR}, R_c)$ . The main consequence is that the merchant's offer becomes *transferable*, i.e., the merchant can no longer establish that the consumer who reveals  $R_c$  and desires to pay for the merchandise is the same one who originally generated OFFER-REQ (and received the corresponding OFFER-REP.)

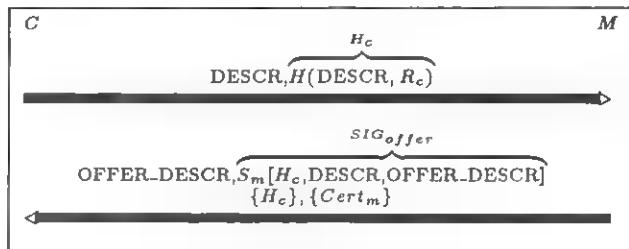


Figure 2: Anonymous PPB protocol without consumer certification

### 3.6 Minor Variations

The PPB method outlined this far can be amended as follows:

1. In the event that the user already has a copy of the merchant's certificate –  $Cert_m$  – OFFER-REQ can be enciphered under the merchant's public key,  $PK_m$ . The resulting secrecy of the message offers protection against snoopers and eavesdroppers.
2. Furthermore, secrecy of the merchant's reply can be obtained if the consumer encloses a secret key (say,  $K_s$ ) in the encrypted OFFER-REQ thus allowing the merchant to use this key for encrypting OFFER-REP.

### 3.7 Security Properties

In summary, the anonymous PPB method has the following security properties:

1. Anonymity of the consumer (while browsing before purchase) with respect to both outsiders and merchants.
2. Authenticity, data integrity and non-repudiation of the offer by the merchant.
3. Unlinkability of multiple requests by the same consumer. This is a particularly important feature

<sup>6</sup> However,  $Cert_c$  is not encrypted or otherwise treated in *iKP*.

which essentially prevents the merchant from recognizing a browsing consumer as someone who has previously made purchases (even if the merchant possesses that consumer's certificate.)

4. Authenticity and non-repudiation of offer request by the consumer (this only becomes apparent when the consumer decides to act upon the offer.)
5. Optionally, privacy from eavesdroppers of merchant-consumer communication.

### 3.8 Practical Considerations

The anonymous PPB method can be applied in both asynchronous (e.g., email-based) and synchronous (e.g., http-based) electronic commerce. In case of the former, the necessary infrastructure already exists, i.e., there are a number of functioning anonymous remailers and several public key certification schemes (even an informal one like PGP's can be used.) Electronic commerce via synchronous communication is currently gathering momentum and is widely believed to be the way of the future. The increasing popularity of WWW-based commerce is a case in point. The only missing ingredient is an http anonymizer – a relay that takes incoming http requests and, acting as a proxy, connects them to desired points of interest while preserving the anonymity of the actual incoming end-points. (Of course, the replies must be mapped back.) We observe that much of this functionality is already provided by firewalls.

## 4 Delivery of Electronic Merchandise

We now consider the issue of consumer anonymity after the payment is already made. If the purchased merchandise is of electronic nature and the consumer has remained anonymous with respect to the merchant during PPB and subsequent payment, there is no reason to give up on anonymity during merchandise delivery stage. However, the anonymous PPB method as described in Section 3 terminates with the consumer revealing his identity upon initiating the payment. This obviously contradicts our goal. For this reason we assume that the PPB method is amended in either of the two ways described in Section 3.5. (In other words, either consumer certificates are not used at all or a trusted third party takes care of asserting the consumer's identity.)

### 4.1 Assumptions and Preliminaries

Prerequisites are almost identical to those for anonymous PPB, i.e., public key infrastructure (for merchants only) and a facility for general-purpose anonymous communication.

The present method commences when the consumer decides to purchase the merchandise based on a previous bid/offer obtained from a run of the PPB protocol. Although it can, in principle, take place concurrently, we assume that payment takes place before the delivery of goods. The particulars of the payment process are out of the scope of this paper. (See [4] or [14] for examples of secure electronic payment protocols and scenarios.)

#### 4.1.1 Step 1

The protocol begins whenever the consumer is ready to take delivery of the merchandise:<sup>7</sup>

- Consumer generates another random number  $R_d$  and computes  $H(R_d)$ .
- Consumer sends to the merchant a COMMIT-REQ message containing:

$$H(R_c), H(R_d)$$

#### 4.1.2 Step 2

- Merchant extracts both  $H(R_c)$  and  $H(R_d)$ ; the latter is stored for future reference.
- Using  $H(R_c)$  merchant locates the corresponding offer in his records. If the offer is no longer valid (e.g., it has been already processed), merchant notifies the consumer accordingly and terminates processing.
- If the offer is still valid, merchant composes and sends to consumer a COMMIT message containing:

$$\underbrace{S_m[\text{COMMIT\_REQ}]}_{\text{SIG}_{\text{commit}}}$$

#### 4.1.3 Step 3

- Consumer receives the COMMIT message and verifies  $\text{SIG}_{\text{commit}}$ . If the signature is invalid, an error message to that effect is sent to merchant.
- If signature is valid, consumer generates and sends to merchant a DELIVERY-REQ message containing:  $R_c$ .

#### 4.1.4 Step 4

- Merchant receives DELIVERY-REQ and extracts  $R_c$ .
- Merchant computes  $H(R_c)$  and compares to  $H(R_c)$  stored in the appropriate transaction record. If there is a mismatch, an error message to that effect is sent to consumer.
- If  $H(R_c)$  values match, merchant composes and sends to consumer a DELIVERY message containing:

$$\underbrace{\text{GOODS}, S_m[\text{SIG}_{\text{commit}}, \text{GOODS}]}_{\text{SIG}_{\text{deliver}}}$$

#### 4.1.5 Step 5

- Consumer receives DELIVERY and obtains GOODS
- Using  $PK_m$  and  $\text{SIG}_{\text{commit}}$  (received in Step 3) consumer verifies  $\text{SIG}_{\text{deliver}}$ . If  $\text{SIG}_{\text{deliver}}$  is invalid an error message to that effect is sent to merchant.

<sup>7</sup>Note that the consumer is assumed to retain  $R_c$  and  $\text{SIG}_{\text{offer}}$  from PPB above.

- Otherwise, consumer sends to merchant a TERMINATE message containing:  $R_d$ .

#### 4.1.6 Step 6

- Finally, merchant receives TERMINATE, extracts  $R_d$ , computes  $H(R_d)$  and compares it with  $H(R_d)$  received in COMMIT-REQ (see step 2.) If they match, the transaction is terminated. Otherwise, an error message is sent to consumer (along with the re-transmission of DELIVERY.)

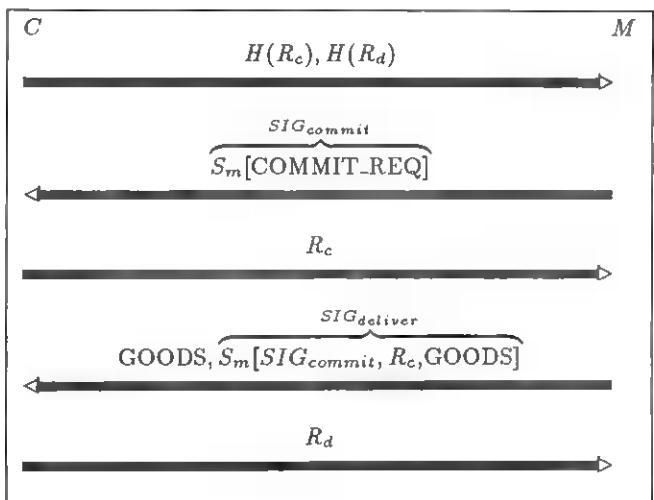


Figure 3: Anonymous EMD protocol

#### 4.2 Dispute Resolution

The anonymous EMD method provides protection against dishonest behavior by either merchants or consumers involved. Potential cases of cheating and disputes are addressed below. All cases require intervention of a mutually trusted off-line authority that we refer to as COURT.

(Some forms of disputes are not addressable in a protocol format. For example, any dispute over the nature and content of GOODS is most likely to be handled on a case-by-case basis.)

While dispute resolution is likely to take place off-line, it is expected that consumer will remain anonymous with respect to merchant. However, consumer may be required to reveal his identity to COURT.

At the end of a successful transaction both consumer and merchant must have the following in their possession:

$$R_c, R_d, \text{SIG}_{\text{commit}}, \text{SIG}_{\text{offer}}, H(R_c), H(R_d)$$

In case of a dispute, the following procedure takes place:

1. Consumer is asked to produce a valid  $\text{SIG}_{\text{offer}}$ 
  - (a) No valid  $\text{SIG}_{\text{offer}}$ ; merchant prevails.
  - (b) Valid  $\text{SIG}_{\text{offer}}$ ; continue with (2).

2. Merchant is asked to provide  $R_c$ .
  - (a) Merchant can not produce the correct  $R_c$ ; continue with (3).
  - (b) Correct  $R_c$ ; continue with (4)
3. Consumer is asked to produce  $R_c$ .
  - (a) Correct  $R_c$ ; consumer prevails (protocol can be re-run.)
  - (b) Incorrect  $R_c$ ; merchant prevails.
4. Consumer is asked to produce  $R_c$ .
  - (a) Correct  $R_c$ ; continue with (5).
  - (b) Incorrect  $R_c$ ; merchant prevails.
5. Consumer is asked to produce a valid  $SIG_{commit}$ .
  - (a) No valid  $SIG_{commit}$ ; merchant prevails.
  - (b) Valid  $SIG_{commit}$ ; continue with (6).
6. Merchant is asked to produce  $R_d$ .
  - (a) Correct  $R_d$ ; merchant prevails.
  - (b) Incorrect  $R_d$ ; consumer prevails (merchant is ordered to send a DELIVERY message to consumer and consumer is ordered to reply with a TERMINATE message; the latter containing a valid  $R_d$ .)

Unlike merchant's other signatures,  $SIG_{deliver}$  is not used in dispute resolution. This is because consumer can always deny having received it. The purpose of  $SIG_{deliver}$  is to provide both origin authentication and non-repudiation of GOODS in the DELIVERY message. Moreover,  $SIG_{deliver}$  can prove very useful in the event of a product misrepresentation dispute (which is out of scope of this paper.)

## 5 Streamlining Anonymous PPB and EMD

The two methods described above are presented independently in order to emphasize their different goals. However, it is certainly possible (and desirable) to combine the two into a unified protocol (as depicted in Figure 4.) The present protocol combines OFFER\_REQ and COMMIT\_REQ messages.

The OFFER REP and COMMIT REP are similarly collapsed into a single message flow. This saves one signature operation for the merchant.

### 5.1 Related Work

There has been a lot of research activity in anonymity in the context of electronic cash and other forms of payments. Anonymity, as considered in this paper, has received far less attention. A notable exception is the NetBill project at CMU [16, 17, 18].

NetBill is an all-around electronic commerce system. It bundles browsing, bidding, payment, delivery and other services. It also includes some provisions for anonymity. This paper, in contrast, considered consumer anonymity as a separate feature, independent of (or orthogonal to)

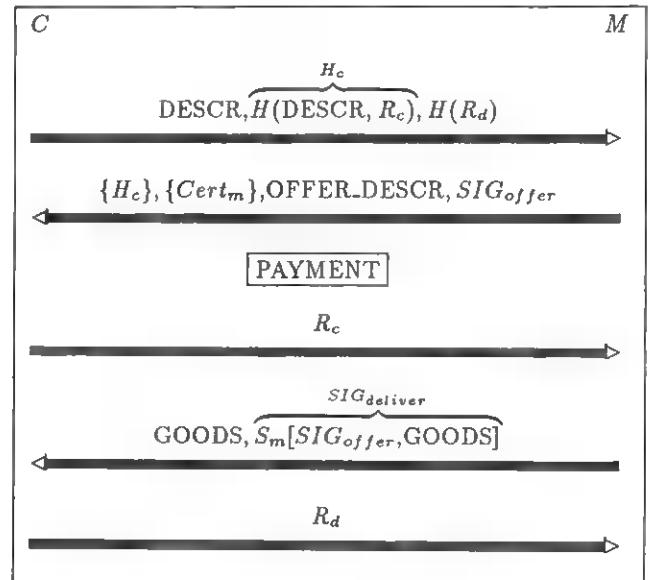


Figure 4: Combined protocol

the particular payment mechanism (NetBill, iKP, Ecash, SET, etc.) and payment type (cash, credit, check, etc.)

Furthermore, since NetBill is built on top of Kerberos, it inherently requires a trusted third party, something we consider only in case of disputes.

NetBill provides two forms of customer anonymity: one-time (per transaction) or multiple use (per merchant). In either case, the customer is not free to choose his own aliases; every single alias must be obtained from a *Pseudonym-Granting Server* (PGS). This requires the disclosure (to the PGS) of the customer's true identity. Consequently, the level of anonymity in NetBill is weaker than that described in this paper.

## 6 Future Work

This paper concentrated on the motivating factors and mechanisms for maintaining consumer anonymity with respect to merchants. While the motivation for the opposite is less apparent, some forms of commerce (auctioning, open bidding) call for merchant anonymity.

A simple (but not always practical) approach is for the consumer to post OFFER\_REQ on an anonymous bulletin board, e.g. a USENET newsgroup and solicit bids. If the consumer trusts COURT or another trusted third party to properly certify the merchants' public keys in an anonymous way (see [2]), the protocols can be run almost exactly as described above, but without the consumer knowing the merchant's identity.

If mutual anonymity is to be attained without the involvement of a trusted third party, we need to look into protocols for anonymous fair exchange [1].

## 7 Conclusions

In this paper we discussed the issue of consumer anonymity in the setting of secure electronic commerce. More specifically, we concentrated on consumer anonymity during *Pre-Purchase Browsing* (PPB) and *Electronic Merchandise Delivery* (EMD). We argued that anonymity does not imply lack of recourse in cases of dishonest merchants. Simple, yet secure, methods for anonymous pre-purchase browsing and anonymous electronic merchandise delivery have been developed. Both methods provide mechanisms for disambiguating inconsistencies and arbitrating disputes.

We have also shown that consumer anonymity in activities surrounding electronic payment (PPB, EMD) does not require any more security infrastructure to be in place than that already necessitated by the state-of-the-art electronic payment systems.

## 8 Acknowledgements

The authors would like to thank Michael Waidner, Michael Steiner, Ari Medvinsky, Cliff Neuman and the anonymous referees for their helpful comments.

## References

- [1] Holger Buerk, Andreas Pfitzmann, "Digital Payment Systems Enabling Security and Unobservability," *Computers & Security* 8/5 (1989) pp. 399-416.
- [2] Ralf Hauser, "Using the Internet to decrease Software Piracy - on Anonymous Receipts, Anonymous ID Cards and Anonymous Vouchers," INET'95, Annual Conference of the Internet Society, Honolulu, HI, June 1995.
- [3] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik and M. Waidner, "iKP - A Family of Secure Electronic Payment Protocols", Usenix Electronic Commerce Workshop, July 1995.
- [4] M. Linehan and G. Tsudik, "Internet Keyed Payments Protocol (iKP)", Internet Draft, "draft-tsudik-ikp-00.txt", June 1995.
- [5] D. Chaum, "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability," *Journal of Cryptology*, 1/1, 1988, pp. 65-75.
- [6] L. Cottrell, "Mixmaster and Remailer Attacks," <http://obscura.com/~loki/remailer-essay.html>.
- [7] J. Linn, "Privacy Enhancement for Internet Electronic Mail — Part I: Message Encryption and Authentication Procedures," RFC 1421, Feb 1993.
- [8] T. May, "Crypto Anarchy and Virtual communities," *Internet Security Journal*, April 1995.
- [9] A. Pfitzmann and M. Waidner, "Networks Without User Observability—design options," *Computers & Security*, 6/2 1987, pp. 158-166.
- [10] P. Zimmerman, "PGP User's Guide", included in PGP distribution 2.6i, October 1994.
- [11] C. Gulcu and G. Tsudik, "Mixing E-mail with BABEL", *1996 Symposium on Network and Distributed System Security*, February 1996.
- [12] D. Chaum, A. Fiat and M. Naor, "Untraceable Electronic Cash", *Proceedings of Crypto'88*, August 1988.
- [13] J. Helsingius, "Penet Anonymous Remailing Service", Information available from [anon@penet.fi](mailto:anon@penet.fi); (set subject field to HELP.)
- [14] MasterCard International, "SET: Secure Electronic Transactions", see: <http://www.mastercard.com/>.
- [15] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Communications of the ACM*, vol. 24, no. 2, February 1981.
- [16] M. Sirbu and D. Tygar, "NetBill: An Internet Commerce System Optimized for Network Delivered Services," *IEEE CompCon Conference*, March 1995.
- [17] B. Cox, D. Tygar and M. Sirbu, "NetBill Security and Transaction Protocol", *Usenix Electronic Commerce Workshop*, July 1995.
- [18] B. Cox, "NetBill Security and Transaction Protocol", *CMU TR-1994-8 (MSc Thesis)*, August 1994. (<http://www.ini.cmu.edu/NETBILL/pubs.html>)
- [19] Community ConneXion, "The Anonymizer", (<http://www.anonymizer.com/>)



# Market-Based Negotiation for Digital Library Services

Tracy Mullen and Michael P. Wellman

*Artificial Intelligence Laboratory  
University of Michigan  
1101 Beal Avenue  
Ann Arbor, MI 48109-2110 USA  
[{mullen, wellman}@umich.edu](mailto:{mullen, wellman}@umich.edu)*

## Abstract

The University of Michigan Digital Library is a large-scale confederation of software agents, providing library content and services to users and each other within a distributed network environment. Allocation of resources and activities to the various agents is determined through a market-based negotiation process, where agents tender offers to buy or sell services, basic resources, and other information goods for specified prices. Generalized auction modules resolve these offers into deals among agents. Viewing each agent as an information consumer or entrepreneur, the digital library as a whole constitutes a virtual economy of information goods and services.

**Topics:** Electronic libraries, negotiations

## Introduction

The University of Michigan Digital Library (UMDL) project (Atkins *et al.* 1996) is a large-scale, multidisciplinary effort to design and build a flexible, scalable infrastructure for rendering library services in a digital networked environment. The aim of UMDL is to provide users with a wealth of information sources and library services, and to provide developers with a medium for producing and delivering new services and collections. Because we cannot anticipate all of the potential services offered within the digital library, flexibility, openness, and extensibility are paramount criteria in design of the system. The UMDL addresses these criteria by providing infrastructure to coordinate the activities of the individual system modules, or *agents*, that perform the actual library services in the system.

In this paper, we first describe UMDL's agent-based infrastructure, its commerce framework, and the key role of commerce in the system. Second, we describe in further detail how the UMDL framework supports a wide range of negotiation mechanisms. Finally, we describe a simple scenario built in

UMDL, illustrating one pattern of negotiation supported by the system. The scenario is intended to be representative (albeit not necessarily realistic in detail) of the sorts of interactions prevalent in a digital library, and thus serves as a proof-of-concept for the underlying approach.

## UMDL Infrastructure

The UMDL architecture realizes a model of the digital library intermediate between a completely open environment (e.g., the world-wide web), and an entirely self-contained system (e.g., proprietary services that impose central control over collection access). Indeed, we can view UMDL as a digital library *intranet*, not in the physical sense of operating on a LAN or WAN, but in the organizational sense of providing a *information services network* which furthers an organization's goals and policies. The openness of the system allows users to benefit from the wide variety of services and collections potentially available, while the controls of the architecture provide structure to manage choices and connections among the diverse modules participating in the system.

To work at this intermediate level, the UMDL infrastructure mediates interactions among *information agents*. The main responsibility of this infrastructure is to facilitate the performance of services by some agents on behalf of others. This facilitation includes disseminating information about the availability and demand for services, and arbitrating connections among the providers and recipients of services.

Openness alone is not enough to gain participation by potential providers of information services, especially when interacting with the library infrastructure incurs some overhead. External agent developers (as well as internal library users, for that matter) will not participate unless they can identify some benefit not already available from the broader internet environment. Such benefits include the facilitation of connections mentioned above, but can also

include more direct compensation, brokered through the system, acquired in exchange for the services or resources delivered through the digital library. Thus, the role of the UMDL infrastructure extends to determining what services are performed and under what terms, and in some cases monitoring and enforcing these terms.

Within the context of such an infrastructure, we can also address internal digital library objectives, such as allocating user, library, or network resources efficiently. For example, one type of library resource that may need to be internally allocated is access to site-licensed information. While a university site license may entitle all Michigan students to access certain information, only a limited number will be able to do so at one time.

### Market-Based Resource Allocation

In a world of unlimited resources, users could avail themselves of all potentially beneficial services at their maximal levels of quality. However, in any real digital library, resources—including raw computational resources, human attention, access to intellectual property—ultimately have some cost.<sup>1</sup> Indeed, in even a moderately scoped digital library, there is potentially unbounded demand for computational resources. For example, any amount of preprocessing of data in the collections—such as indexing, meta-data gathering, or caching—might improve the response of the system to subsequent user requests. Given only finite resources, however, we cannot take advantage of all such opportunities. We also cannot generally try every method for accomplishing a given task, but rather must choose among those available based on resource requirements and prospects for success. We therefore seek principled methods for expressing operating tradeoffs and allocating computational resources toward their maximal expected benefit. Moreover, we require that these methods be sufficiently flexible to adapt to patterns of usage that evolve during the operation of the digital library.

In approaching this resource allocation problem, we treat the alternative information services as competing economic activities. Given a measure of priorities over the end-user services provided, the various agents effectively compete to provide the highest level of service using the minimal computational resources. One central capability of agents is thus to be able to reach agreements on suitable compensa-

<sup>1</sup> Costs of some resources reflect the denial of opportunity to deploy them in alternate activities within the library. Costs of others reflect those incurred outside the library system.

tion. The goal is to achieve an efficient overall allocation of resources towards the optimal provision of services to users.

To organize the processing activities within an economic framework, we view interactions between the agents as supplier-producer relationships, where each agent produces value-added information products from the input products provided by others. Agents dynamically connect with each other as opportunities arise for mutually beneficial exchanges. The collections, represented by *collection interface agents* (CIAs), provide some ultimate “raw materials” in this process. Library end users, represented by *user interface agents* (UIAs) are the ultimate consumers of the “finished goods”. *Mediator* agents (“middlemen”) bridge the gap by bringing to bear knowledge, processing, storage, or other computational resources to improve in some way the expected value of the information as it passes along the chain from agent to agent.

The particular set of goods and the mechanisms by which they are negotiated and exchanged in the UMDL constitute the system’s *market configuration*. In a dynamic and open environment, such as UMDL, it will be impossible to fix the configuration in advance. Moreover, finding an appropriate mechanism for determining exchange terms may vary depending on the current configuration, the good to be exchanged as well as other contextual factors. Therefore, to provide sufficient flexibility, UMDL must support dynamic market configurations through a extensible means of describing goods and wide variety of negotiation and payment mechanisms.

### UMDL Negotiation Mechanisms

A negotiation and exchange facility for information goods and services must potentially support a wide variety of market types. For example, if one were to sell an electronic version of a magazine, it could potentially be sold in many different ways, such as:

- by subscription, individual issue, or individual article,
- to individuals, libraries, or other defined group,
- prior to paper publication, immediate on publication, or with delay,
- with or without license to further distribute.

The magazine might also be bundled with related services such as searching or notification of new articles.

Each of these different magazine goods might be offered under distinct sorts of negotiation procedures.

For example, a single issue of the magazine might be offered by the supplier for a set price, as at a typical newsstand. The negotiation in this case degenerates to a take-it-or-leave-it proposition. On the other hand, the cost of search services might depend upon current congestion levels in the digital library system—when the system is more heavily loaded, a more timely response would cost more. In this case, the user may wish to specify a variable willingness to pay based on the immediacy of response. The system could then match the buyer's demand profile against current congestion profiles to determine an appropriate level of service.

Similarly, even the same magazine good could be exchanged via different negotiation mechanisms in different contexts. For example, a back issue of a magazine would typically be bought from the publisher for a set price, whereas one might haggle for it at a flea market. Thus it is the combination of the good and the negotiation mechanism which determines any market.

Each of the different good descriptions, negotiation and payment procedures may have advantages and disadvantages for the various types of information goods and services. While analyzing these issues is an important topic for research in digital libraries, we do not believe it appropriate for UMDL to dictate a particular solution. Instead, we provide generic facilities for defining these properties and rely on context and policy decisions to determine when and where they should be used.

The remainder of this section focuses on describing our generic negotiation specification approach. We use auction protocols as a framework to describe various mediated negotiations. However, there is no reason to believe that this approach would not extend to other types of negotiation methods as well.

Section presents a very simple, but representative, scenario—implemented in UMDL—where typical digital library goods and services are exchanged using one particular class of market mechanisms.

### Market Facilitators

Within a multiagent environment, system agents specializing in brokering connections among other agents are often called *facilitators* (Genesereth & Ketchpel 1994). In the UMDL, *market facilitators*, or *auctions*, serve to determine which agents provide services to which others, under what terms. UMDL auctions operate by collecting offers and determining agreements consistent with those offers. For example, one simple kind of auction collects bids and settles them by finding a single price that clears the market, while others may perform a complicated

matching procedure.

The negotiation protocol we have defined in the first UMDL prototype is based on a simple kind of market facilitator. There is one auction agent for each *good*, where a good may denote, for example, the delivery of digital objects, the provision of translation services, or other library product. Each auction accepts bids from agents interested in buying or selling that good. Bids include a demand schedule specifying the amount (generally, some quantity or quality measure) the agent is willing to buy or sell at various prices. The auction's task is to find a price such that supply balances demand, that is, the expressed amount agents are willing to sell in aggregate matches the amount other agents are willing to buy.

During the bidding process, market facilitators may also notify bidders of tentative prices, called *price quotes*, for use in comparison shopping or coordinating other activities. After a market clears, the auction may invoke some transaction execution services, depending on the nature of the exchange being mediated. For instance, we are currently planning to integrate UMDL with the NetBill payment mechanism (Sirbu & Tygar 1995) to exchange monetary fees for digital objects. As with negotiation mechanisms, the ideal payment mechanism may depend on many context-specific factors (MacKie-Mason & White 1996), and so it may be advantageous to support a generic payment interface (Ketchpel *et al.* 1996).

### Specifying Auctions

We define auctions in UMDL via a straightforward auction specification language. This language serves two purposes. One is to organize the software implementation, so that components realizing particular auction features can be combined and reused. The second is that agents can refer to auction descriptions within the UMDL agent communication protocol to create particular auctions or to inquire about their characteristics.

In designing this specification language, we have attempted to exploit regularities common across different auctions. Building on other auction classification schemes (Engelbrecht-Wiggans, Shubik, & Stark 1983; Friedman & Rust 1993; Kagel & Roth 1995), we culled a working set of parameters that allow us to describe any of the major auction types.<sup>2</sup> See Table 1 for a partial set of these parameters and

<sup>2</sup>This effort was performed in conjunction with the Michigan Internet AuctionBot project, with assistance from Peter Wurman and William Walsh, (<http://auction.eecs.umich.edu/>).

their description. Recall that we do not impose a single auction mechanism in UMDL, instead supporting a variety of types, including those found in various survey articles (McAfee & McMillan 1987; Milgrom 1989; Smith 1989).

The central defining parameter of an auction is its clearing policy, which specifies the rules for determining the prices and allocation resulting from a set of bids. The economic properties of the auction—optimal bidding strategies, efficiency, welfare distribution—typically depend critically on these rules. For example, if a single unit of a good is being auctioned and incentive compatibility for buyers (i.e., a buyer's optimal strategy is to bid the true valuation of the item) is paramount, then a second-price (Vickrey) auction is appropriate. If there are multiple units and price discrimination (i.e., charging different buyers different prices) is acceptable, then the generalized Vickrey procedure described by Varian (Varian 1995) might be a good choice. Under alternate circumstances, yet other rules would yield the best balance of desiderata. For example, some key criteria for choosing auction types within UMDL are fostering market configurations that are incentive compatible and individually rational. The incentive compatibility criterion means that the protocol encourages rational agents to report their true values in bids. Individual rationality dictates that outside agents (and inside users) gain some advantage to participating in the system.

In our model, auctions can operate continually, clearing multiple times before they finally close, if ever. For example, one might specify that an auction should have a *clearing interval* of every three seconds, while the *final clear* should not occur until January 1, 2000. To get a better idea of how this works, consider the common English auction, where an auctioneer incrementally raises the price until there are no more takers. Such an auction would be specified in terms of our parameters as in Table 2.

Another kind of English auction, commonly seen on the World-Wide Web,<sup>3</sup> does not set the price quote according to this fixed-increment scheme. Instead, participants bid asynchronously, and the current highest offer is used as the price quote. This kind of English auction can easily be specified by changing the *price quote policy* in Table 2 to *first price*, (i.e., price quote equals current highest buyer's offer).

It is possible to specify unnamed auctions by filling in parameters in various ways. Of course, certain

combinations of parameters would not be valid or might not make sense (such as having a price quote interval longer than the final clearing time). Such validity constraints must be enforced by the UMDL process that creates auctions from these parametric descriptions.

### Creating and Finding an Auction

UMDL auctions may be created for any of several reasons. Shortages in basic computational resources (bandwidth, storage, processing) may cause the UMDL system to create auctions in those resources. Or, new markets may spring up in response to topical interests, for example, during hurricane season there may be a special market for weather services that track hurricanes.

One of the more common events triggering a new auction is when an information provider joins UMDL and wants to sell its product. To start the auction, the provider must first describe its product using UMDL's *goods description language*. The advantage of using a structured description language for goods is that agents may participate in multiple, related auctions covering the goods they are interested in buying or selling. For example, suppose the information provider, Sports-R-Us, is selling a magazine called *Teen Sports*. Based on specified features of this magazine, it might be determined automatically that there are several available auctions for exchanging this good, such as:

1. **Teen Sports auctions** *Teen Sports* magazine only
2. **Sports-R-Us Magazines auctions** all Sports-R-Us magazines
3. **Teen Magazines auctions** all kinds of teen magazines
4. **Sports Magazines auctions** all kinds of sports magazines
5. **Magazines auctions** all kinds of magazines

Sports-R-Us might choose any or all of these auctions to participate in. While they would probably create the *Teen Sports* and *Sports-R-Us Magazines* auctions, the others may be created by arbitrary agents. In deciding whether to join these auctions, Sports-R-Us would want to examine how they operated, by inspecting their specifications.

Suppose Sports-R-Us decides to participate only in a *Teen Sports Auction*. Since this auction does not exist already, they will need to create it by specifying the desired auction parameters. In this case, they would probably specify that this auction allows

<sup>3</sup>For pointers to auction-related web material, see <http://auction.eecs.umich.edu/other-auctions.html>.

<i>Parameter</i>	<i>Description</i>
<b>Good Divisibility</b>	Are good units discrete or continuous?
<b># Buyers</b>	Number of buyers (one or many)
<b># Sellers</b>	Number of sellers (one or many)
<b>Information Revelation</b>	Amount and types of information publicly available
<b>Price Quote Policy</b>	Method for calculating the price(s) reported in price quotes
<b>Price Quote Interval</b>	Interval (frequency) with which price quotes are posted
<b>Clearing Policy</b>	Method for calculating the resulting prices and allocations
<b>Clearing Interval</b>	Schedule (frequency) for clearing the auction
<b>Final Clear</b>	Time auction is terminated
<b>Tie Breaking</b>	Criteria for resolving tied bids

Table 1: Auction specification parameters.

only one seller, themselves. Once they have specified the auction type, an appropriate market facilitator agent, can be instantiated. The new auction advertises its services with the UMDL *Registrar*, a database agent that acts as an initial contact point for finding other agents in UMDL.<sup>4</sup>

On the other side of the economy, end users wishing to purchase *Teen Sports* interact with UMDL through their UIAs. UIAs may in turn contact specialized intermediary or broker agents that can help locate, recommend, or purchase products. However, in the simplest case, a UIA can locate the appropriate auction by querying the Registrar, receiving back pointers to auctions where that good can be bought. This process is depicted in Figure 1.

### Auction Negotiation

Once an agent has located an appropriate auction, it can make an *offer* to buy or sell that good. The exact form of the offer depends both on the kind of auction as well as the preferences of the bidder agent. Table 3 shows an example of two offers to two different types of auctions (for different kinds of goods). The English auction is as specified above; the *Walrasian* auction is designed for multilateral exchange of continuous commodities, as described in Table 4.

An offer needs to specify both the price offered to buy or sell the good, and any qualifying information, such as expiration date. In the case of the

Walrasian auction, the price component is variable, allowing agents to associate different prices with different quantities or quality levels. The *bid schedule* specified in the offer is essentially the agent's demand curve for the good. Offers may also include requests for notification of various auction events, such as new price quotes or when a bid expires.

The auction's clearing policy determines the allocation of goods and terms of exchange as a function of the offers it has received. For English auctions, the winner is the highest bidder at the last bid price. For the Walrasian auction, the clearing price is determined to balance buy and sell quantities, with each agent exchanging according to their expressed demand at that clearing price.

UMDL assumes all offers to be binding, subject to the commitment rules of the auction type. We do not currently support conditional bids, where bidders make offers contingent on sale or purchase of other goods. Instead, we deem it the bidders' responsibility to manage its interdependent bids by monitoring price quotes for its active auctions. Thus when an auction clears, any offers that have not expired and have not been retracted are considered valid offers. Pending integration of digital payment mechanisms, the only remaining task of the auction is to notify the two parties. In subsequent development, we expect the auction to initiate transactions using payment schemes agreeable to the parties involved.

<sup>4</sup>In practice, a agent may need to contact a separate *Auction Manager* rather than the Registrar. However, the two can be thought of as components of a single UMDL registry function.

Parameter	Values	Explanation
Good Divisibility	no	good exchanged as a single unit
# Buyers/# Sellers	many/one	
Information Revelation	bids transactions winner	most information about the process is publicly available, including bids, final clearing price, and winner identity
Price Quote Policy	+delta prices	auctioneer incrementally raises price
Price Quote Interval	immediate	price quote announced whenever it changes
Clearing Policy	first price	clearing price equals highest buyer's offer
Clearing Interval	inactivity	auction clears after idle period
Final Clear	inactivity	same as above
Tie Breaking	arrival time	earliest bid wins

Table 2: English auction specification: Price raised by auctioneer.

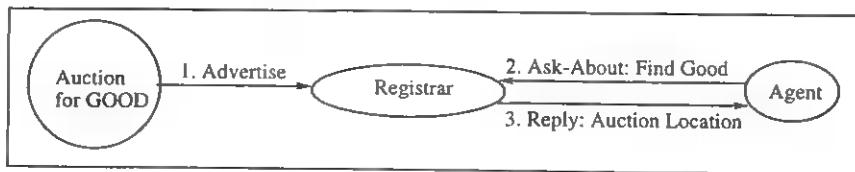


Figure 1: Finding an auction.

## Scenario

To test and illustrate our negotiation mechanisms, we developed a simple market configuration reflecting a typical set of exchanges that might be expected in a digital library. We start with a UIA desiring to purchase various digital library goods and services, depending on its needs, budget, and prices. At the other end, a CIA wants to sell access to its collections, and acquire the system resources necessary to provide this service in UMDS. The "system resources" we consider here are really an amalgam of network and computational resources, which are bundled as a single good to simplify the initial scenario.

In the particular scenario we have constructed, depicted in Figure 2, two types of mediator agents can produce services that enhance the value of the CIA's product to the UIA. In particular, a *Notify* agent provides notification services, sending messages to the user when designated new material has been placed in a collection. A *Search* agent supplies search facilities over one or more collections. Both agents require some raw information resources, collection access and system resources, to produce their value-added services. As shown in Figure 2, the agents negotiate exchanges of relevant goods through their

respective auctions, where arrows indicate the flow of goods.

The two mediator producers compete for demand of the UIA, and to acquire available resources. The UIA would likely consider the two services partly substitutable, as some amount of search can replace the service of the Notify agent, and vice versa. As the user's relative preference for notification compared to search increases, we would expect more of the system's resources devoted to the former rather than the latter.

## Configuration

In general, configuring an information economy entails specifying the goods, the agents, and the market mechanisms. Since our focus was on market integration, we restricted the number and complexity of the goods and agents involved. As mentioned above, we abstracted all computational and network resources into a single "system resources" good. Similarly, we chose to measure other goods in one-dimensional quantity units. For example, we assume a scale for level of notification service, rather than support separate negotiation for the variety of parameterizations of the service (e.g., notify twice a day for three weeks of the event types).

Auction Type	Offer Description	
English <i>many buyers one seller indivisible good</i>	Bid Schedule:	buy 1 unit at \$5
	Notify:	price quotes
	Expiration:	at next clearing
Walrasian <i>many buyers many sellers divisible good</i>	Bid Schedule:	buy 5.3 units at \$1 buy 1 unit at \$5 sell 1.5 units at \$10
	Bid Interpolation:	linear interpolation between bid-schedule points
	Bid Extrapolation:	none
	Notify:	price quotes
	Expiration:	at next clearing

Table 3: Examples of offer formats for two auction types.

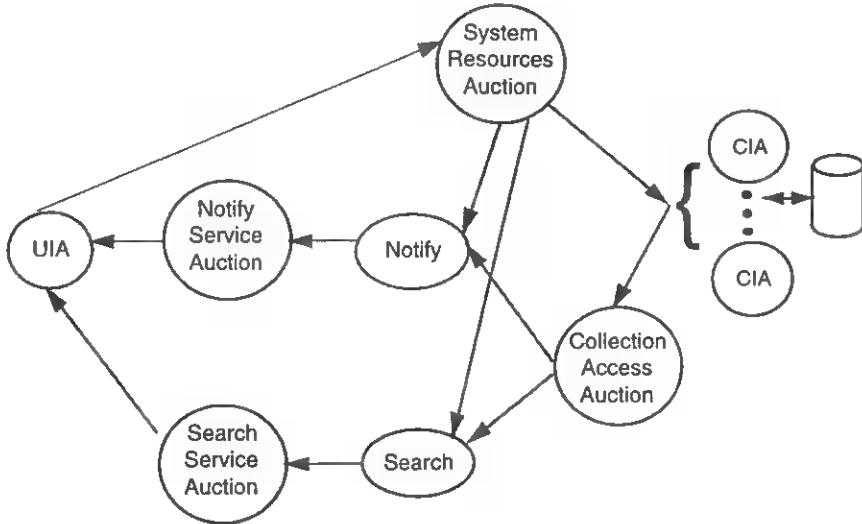


Figure 2: A simple UMDL negotiation scenario.

We designed our market configuration for the scenario within the general equilibrium framework employed in previous work (Mullen & Wellman 1995; Wellman 1993). In microeconomic theory, *general equilibrium* refers to the situation where multiple interconnected markets reach an overall price equilibrium. Clearing a single market in isolation is called *partial equilibrium*. Because a realistic digital library must simultaneously allocate numerous, highly interrelated resources, we deem general equilibrium the more salient analytical concept for this environment.

Adopting the formal general equilibrium framework, along with the competitive mechanism, resulted in three main advantages. First, we were able to use tools from economic theory to build the econ-

omy and verify the results. Second, the assumption of competitive behavior simplifies the design of agents. Competitive agents take prices as given—ignoring any market power they might possess, and thereby eliminating the need to reason strategically about other agents’ bidding behavior. Moreover, competitive equilibria are guaranteed to be Pareto optimal. Third, only one kind of market mechanism, the Walrasian auction, is required.

Of course, the assumptions entailed by the general-equilibrium framework are also quite restrictive, and any analytical results that depend on these assumptions do not automatically transfer to the general UMDL context. Note however, that the UMDL negotiation mechanisms themselves are *not* inherently limited to competitive agents, Walrasian auctions,

equilibrium-based transaction, or other simplifying features of the scenario presented here.

**Consumers** The general-equilibrium framework posits two basic types of agents, *consumers* and *producers*. The consumers in UMDL are the UIAs. Each UIA in the library is endowed with an allocation of system resources, which it can sell to the system in order to raise funds to purchase desirable services.<sup>5</sup> The consumer decides how much of each available service—in this case notify or search—to purchase depending on how much it values the service, the service’s price, and how much it can afford. We represent user preferences by the CES (constant elasticity of substitution) utility function, which has the following form. The choice variables  $x_{sr}$ ,  $x_n$ , and  $x_s$ , respectively, denote consumption of system resources, notification service, and search service.

$$U(x_{sr}, x_n, x_s) = (\alpha_{sr}x_{sr})^\rho + (\alpha_n x_n)^\rho + (\alpha_s x_s)^\rho. \quad (1)$$

The CES utility function provides analytic convenience and a reasonably straightforward way of expressing preferences between goods. The coefficients  $\alpha_i$  weight the consumer’s preference for good  $i$ , and the  $\rho$  parameter controls the degree of substitutability among goods. To run the model, we arbitrarily set  $\rho = 1/2$  and decided that the user preferred the search service twice as much as the notify service, ( $\alpha_s = 2\alpha_n$ ). The users also derived a small amount of utility from keeping system resources for a later time, so  $\alpha_{sr}$  was set to a small value.

The consumer’s demand function,  $x(p)$ , is derived by maximizing the above CES utility function subject to the budget constraint, given prices  $p$ . This problem has a closed-form solution, used by the UIA in generating offers for the different services desired. As shown in Table 3, the bid schedule for a UMDL Walrasian auction is specified as a series of price-quantity pairs, with linear interpolation to approximate the complete demand curve. By concentrating on points near the current price, a UIA can achieve reasonable approximation of the actual demand curve without having to specify very many points.

**Producers** Producers transform input goods into output goods according to constraints specified by their *technologies*. A competitive producer’s goal is to maximize its profits, given its technology and the current relative prices of their input and output

<sup>5</sup>In the operational UMDL, we plan to endow the UIAs with currency directly, and supply basic computational resources through designated system agents. In the experimental versions of the system we employ a fake currency, which we call “BiblioBucks”.

goods. In UMDL, we view all CIAs and mediators as producer agents.

For this scenario, we used three types of producers. CIAs use system resources to produce access to their collection. We assume the system is congested, so that the cost per unit of system resources provided increases with the load on the system. In other words, the more system resources are required, the more the congestion, the higher the unit cost. We represent this technology by a quadratic-cost production function where the production of collection access,  $y_{ca}$ , for a given consumption of system resources,  $x_{sr}$  is described.

$$x_{sr} = 0.01 * y_{ca}^2 + 0.05 * y_{ca}. \quad (2)$$

Another type of producer, Notify agents, bundle system resources and collection access to produce notification service. Buying one unit of system resources,  $x_{sr}$ , and one unit of collection access,  $x_{ca}$ , provides one unit of notification service,  $y_n$ .

$$y_n = \min(x_{sr}, x_{ca}) \quad (3)$$

Finally, Search agents also bundle system resources and collection access, but produce search service rather than notification.

**Walrasian Auctions** Auctions determine the resource allocations and prices for each good. For this scenario, we only need to use one type of market mechanism, the Walrasian auction, described in Table 4. In particular, we set the *price quote interval* to be every two seconds and the *final clear* time to be two minutes from the starting time.

Offers, described in Table 3, can be submitted to the auction at any time up until the final clearing time. When an agent submits a new offer, that offer will supersede any previous offer. A price quote will be generated every two seconds by doing a hypothetical clear of all currently valid bids. This price quote is reported to each participating agent. Once an agent receives the new price, it may want to reconsider its bids for the other goods it is interested in and send out new offers. For example, if the price of search service changes, then an outstanding offer for notify service (based on how much the agent previously thought it would have to pay for search) will no longer be accurate.

Ideally, the Walrasian auctions clear until there is a simultaneous equilibrium across all of them, i.e., they reach a general equilibrium. One way of determining when the economy has reached a general equilibrium across related markets is when no bidding activity occurs in any of the markets. However,

<i>Parameter</i>	<i>Values</i>
<b>Good Divisibility</b>	yes
<b>#Buyers/#Sellers</b>	many/many
<b>Information Revelation</b>	prices
<b>Price Quote Policy</b>	zero excess demand
<b>Price Quote Interval</b>	interval
<b>Clearing Policy</b>	zero excess demand
<b>Final Clear</b>	date & time
<b>Tie Breaking</b>	arrival time

Table 4: Walrasian Auction

this condition can not easily be assessed in decentralized, dynamic environments. Instead, each auction can be given a time limit before it will execute its final clearing. For our scenario, this time limit could be set long enough so that general equilibrium would be reached. Of course, in the general case, there are no guarantees that a general equilibrium will have been reached when the auction transacts. Depending on the nature of the market, this discrepancy might have negligible impact on the quality of the resource allocation or else it might require that different kinds of bidding protocols and periodic price-adjustment mechanisms be used instead.

At the auction's final clearing, the actual price is determined, as well as all buy/sell exchanges between agents. The terms of each transaction, the price and quantity to be exchanged, is sent to each of the matched agents.

## Results

Within a digital library system, services will constantly be added and removed, reflecting newly available opportunities as well as changing user interests and priorities. Our scenario captures the dynamic flavor of digital library negotiation by running the market for some time with only one service provider, adding a second service while the negotiation is in progress. From the simulation, we see how the market adjusts to a new equilibrium reflecting the modified situation.

Initially, there is only one service—notify—available for the UIA to purchase. With no other choices available, the UIA will spend most of its endowment of system resources to acquire notify services. Since system resources is the numeraire good in this scenario, (i.e., its price is fixed at one), and it acts essentially like money.

As each agent bids at different auctions, prices converge to the competitive equilibrium prices and quantities shown in Figure 3. If the auctions were to clear at this point, the UIA would sell 610 units

of its system resources to purchase 200 units of notify service. In order to produce those 200 units of notify service, the Notify agent would have bought 200 units of system resources and 200 units of collection access. Finally, in order to produce 200 units of collection access, the CIA would require 410 units of system resources, as dictated by its production technology.

However, before the final clearing time for the auctions, a Search agent is dynamically added to the economy. Since system resources and collection access can now be used for search services as well as notify services, the economy readjusts to a new Pareto optimal allocation reflecting the new uses for those resources, as can be seen in Figure 4. Normally prices, as well as allocations, would change, but since the two competing services have the same constant-returns technology in this scenario, prices remain unchanged.

Notice that it is the end user, or UIA, who drives how system resources and collection access will be reapportioned between the service providers. In our case, we designed the UIA to prefer search services twice as much as notify services, at a given price. Figure 4 does indeed verify that, given the same price for the two services, the UIA bought twice as much search service as notify service.

In fact, since this economy was designed within a general equilibrium framework, we were able to use the underlying economic theory to build the scenario and verify our results before ever running the scenario in UMDL. Given the producers' technologies, we applied standard microeconomic identities to derive utility function parameters and endowments consistent with the desired behavior (Mullen & Wellman 1995). Once the economy was specified, we verified the expected competitive equilibrium by simulating the configuration in the WALRAS market-oriented programming environment (Wellman 1993). As expected, the UMDL negotiation mechanism reproduces the result predicted by the theory.

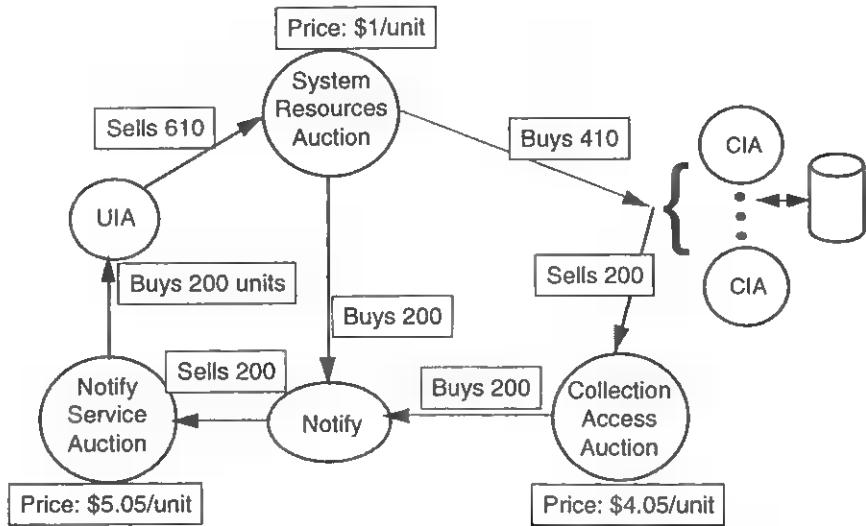


Figure 3: Finding initial equilibrium prices and allocations

## Discussion

The preceding scenario illustrates the basic workings of negotiation within UMDL, and demonstrates the role of market-based negotiation in dynamic resource allocation. Negotiation mechanisms have thus far been implemented in only a subset of UMDL agents; we are working on providing generic negotiation facilities as part of the kernel agent software, and developing specific negotiation strategies for a broader range of agent types.

Market-based approaches have recently been investigated for a variety of problems in distributed resource allocation (Clearwater 1996). In particular, researchers at Columbia and IBM have explored over the past several years a variety of resource-allocation models applicable to distributed network resources (Ferguson *et al.* 1996), and potentially digital libraries. The SIGMA information retrieval system (Karakoulas & Ferguson 1995) is expressly conceived as a computational market, and we expect that the particular SIGMA information agents would be relatively at home in UMDL. Our own prior work studied a computational market model of network information services (Mullen & Wellman 1995), and served as a prototype for the UMDL information service economy discussed in this paper.

Several of the emerging large-scale digital library systems are incorporating payment mechanisms, primarily to support compensation for intellectual property usage. Most of these systems start from agreements negotiated out-of-band, or support a model of seller-specified take-it-or-leave-it pricing. The development of more flexible automated negotiation mechanisms is a relatively neglected problem, one

that we are addressing directly in our work on UMDL.

**Acknowledgments** This work was supported by the NSF/ARPA/NASA Digital Library Initiative. Participants in the UMDL project have contributed to the concepts described here, especially William Birmingham, Edmund Durfee, Wendy Lougee, Jeffrey MacKie-Mason, and William Walsh. We are especially thankful to Steven Ketchpel, William Walsh, Peter Weinstein, and our anonymous referees for comments.

## References

- Atkins, D. E.; Birmingham, W. P.; Durfee, E. H.; Glover, E. J.; Mullen, T.; Rundensteiner, E. A.; Soloway, E.; Vidal, J. M.; Wallace, R.; and Wellman, M. P. 1996. Toward inquiry-based education through interacting software agents. *IEEE Computer* 29(5):69–76.
- Clearwater, S. H., ed. 1996. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific.
- Eatwell, J.; Milgate, M.; and Newman, P., eds. 1989. *Allocation, Information and Markets*. W. W. Norton and Company, Inc.
- Engelbrecht-Wiggans, R.; Shubik, M.; and Stark, R. M., eds. 1983. *Auctions, Bidding and Contracting*. New York University Press.
- Ferguson, D. F.; Nikolaou, C.; Sairamesh, J.; and Yemini, Y. 1996. Economic models for allocating resources in computer systems. In Clearwater (1996).

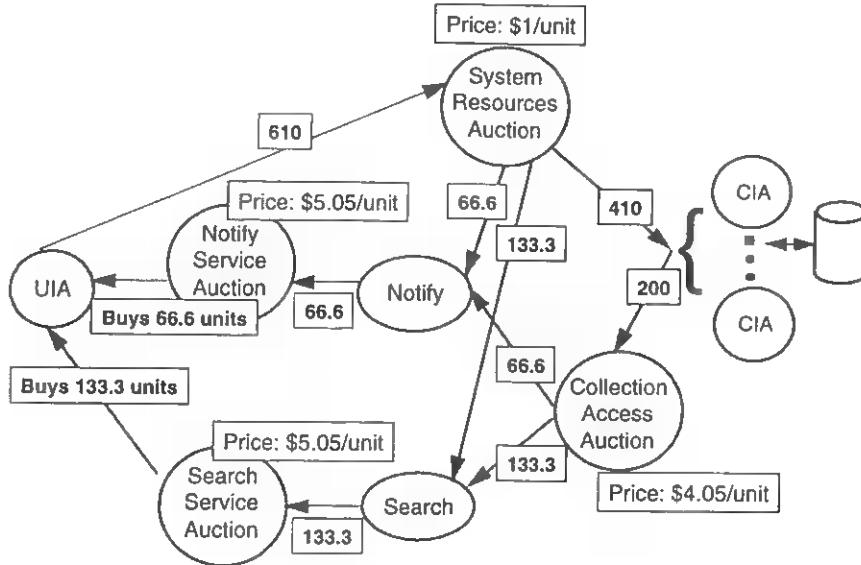


Figure 4: Finding final equilibrium prices and allocations.

Friedman, D., and Rust, J., eds. 1993. *The Double Auction Market*. Addison-Wesley.

Genesereth, M. R., and Ketchpel, S. P. 1994. Software agents. *Communications of the ACM* 37(7).

Kagel, J. H., and Roth, A. E., eds. 1995. *The Handbook of Experimental Economics*. Princeton University Press.

Karakoulias, G. J., and Ferguson, I. A. 1995. A computational market for information filtering in multi-dimensional spaces. In *AAAI Fall Symposium on AI Applications in Knowledge Navigation and Retrieval*.

Ketchpel, S. P.; Garcia-Molina, H.; Paepcke, A.; Hassan, S.; and Cousins, S. 1996. U-PAI: A universal payment application interface. In *Second USENIX Workshop on Electronic Commerce*.

MacKie-Mason, J. K., and White, K. 1996. An axiomatic approach to evaluating and selecting digital payment mechanisms. In *Telecommunications Policy Research Conference*.

McAfee, R. P., and McMillan, J. 1987. Auctions and bidding. *Journal of Economic Literature* 25:699–738.

Milgrom, P. 1989. Auctions and bidding: A primer. *Journal of Economic Perspectives* 3(3):3–22.

Mullen, T., and Wellman, M. P. 1995. A simple computational market for network information services. In *First International Conference on Multiagent Systems*, 283–289.

Sirbu, M., and Tygar, J. D. 1995. NetBill: An internet commerce system optimized for network delivered services. *IEEE Personal Communications* 2(4):34–39.

Smith, V. L. 1989. Auctions. In Eatwell et al. (1989).

Varian, H. R. 1995. Economic mechanism design for computerized agents. In *First USENIX Workshop on Electronic Commerce*.

Wellman, M. P. 1993. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research* 1:1–22.



# Information and interaction in MarketSpace— towards an open agent-based market infrastructure

Joakim Eriksson    Niclas Finne    Sverker Janson

*Swedish Institute of Computer Science  
SICS, Box 1263, S-164 28 Kista, Sweden  
[{joakime,nfi,sverker}@sics.se](mailto:{joakime,nfi,sverker}@sics.se)*

## Abstract

The lack of structure of information and interaction in current web-based electronic commerce makes partial or full automation infeasible. We describe the first steps towards an open agent-based market infrastructure, with well-defined information and interaction models allowing agents to locate relevant market participants, exchange interests, and negotiate deals. We also describe an architecture and a prototype based on these ideas, and illustrate the operation of the prototype in an example scenario.

## 1 Introduction

Current web-based commerce is by and large a replica of real-world commerce. Customers are supposed to visit various electronic storefronts, inspect the digital displays of goods and services, and, perhaps after some comparison shopping, place an order.

Unfortunately they have to do it in person. Attempts at supporting the customer through “intelligent agents” are guaranteed to be in vain, unless the underlying model is changed. It would have to be one bright agent to, using the web as is, find a few good deals on a fridge at nearby resellers, haggle prices and delivery conditions, and then present the three best alternatives. Which minute fraction of the 50000 hits on “refrigerator” in AltaVista is at all relevant? In these, what part of the text is the price, the model, the delivery conditions?

Needless to say, much better models are possible, in which the above would be a very basic service, and one such model will sooner or later supplant the web-based models. The web, and associated technologies such as Java, will remain as the user interface, but a new infrastructure will emerge that is targeted to the task of creating a near-perfect global market.

In this paper, we present work towards such an in-

frastructure, characterized by *openness* and by being based on a paradigm of *interacting agents*.

We would like the market to be open like the web. Nobody should own it. Anyone should be able to enter it. Anyone should be able to offer any kind of service, such as brokering. This is typically not the case with current markets, where all information is owned by the market operator and not available for use by others [1, 2].

A market has to support more activities than just finding the desired product or customer. Well-defined interaction protocols are needed for negotiation. Ideally, these protocols should make sense for human-human, human-agent and agent-agent interaction alike, to make possible any mix of human and automated participants in the market. (For agent-based approaches, see, e.g., Kasbah [3] and MAGMA [4]. For pertaining discussions see [5].)

Note that although the issues of security and payment are strongly emphasized in most works on electronic commerce, they are quite orthogonal here. Any future standards will do.

## Organization of this document

In the remaining sections of this paper, we outline the information and interaction models that support our preliminary MarketSpace infrastructure, describe the design of a MarketSpace server architecture and prototype, and finally offer a few concluding remarks.

## 2 Information and interaction

We will need an information model in which we can describe information about participants and their interests, different products, delivery specification, etc. The model must be flexible in a way that a new product can be described and used, and structured so that the information described is easy to parse

for a program. There must also be an interaction model, setting up rules for how the participants in MarketSpace can negotiate with each other.

In the rest of this section we will introduce our approach to the information and interaction models for MarketSpace.

## Information

Interests are the essential pieces of information in a market. They describe what a participant (person, company, organization, etc) is concerned about, curious about, intends to do or wants. There are several different kind of interests from simple forms of interests like "I am interested in cats" to more complicated like "I want to buy all things needed to build a spacecraft and hire personnel to build it". Since the goal of a market participant is to make deals, an interest can be said to describe a set of potential deals. We will now try to make these concepts a little more precise.

What we will need to know about a *deal* is that it has a type, some participants involved, and a time when it is entered. As basic types we have (the names of) *participants* in the market, which are either human or programmed, and *time points*. The *deal type* includes everything else, such as what is sold, delivery conditions, etc. With each deal type is associated a set of *rôles*, which are the parts played by participants in a deal, such as buyer or seller. A deal assigns participants to all the rôles in its type.

An *interest* is a set of deals. An *expression of interest (eoI)* is a representation of an interest.

## The interaction model

Based on the interests, we define a very simple speech-act style interaction protocol. It consists of *information messages*: ask and tell, and *negotiation messages*: propose, reply, accept, and reject.

*ask(A, B, eoi, id)* A asks B for information on interests, giving an eoi (possibly but not necessarily its own interests) as guidance. Replies can be given by tell.

*tell(A, B, eoi), tell(A, B, eoi, id)* A tells B about some interests (possibly its own). If this is intended as an answer, the question can be referred to by its id.

*propose(A, B, eoi, id)* A initiates negotiation with B, giving an interest as guidance. Replies are given by reply, accept, or reject.

*reply(A, B, id, eoi, id)* A replies to B's latest bid.

The interest given need not have any special relation to the bid. (It is up to the participants involved to assess the progress of the negotiation and interrupt it when necessary.)

*accept(A, B, id, id)* A accepts B's latest bid. This is equivalent to signing a contract. The reply is to accept or reject.

*reject(A, B, id)* A aborts the current negotiation with B.

We are currently exploring conversational styles and idioms using these basic messages, rather than integrating capabilities through complex message types. This will allow some, simpler, participants to have a more naive view of interaction, while more advanced participants can recognize and enjoy the benefits of more elaborate patterns of interaction.

Other agent communication languages include KQML (see, e.g., [6]), which, being focussed on the communication of knowledge, does not offer specific performatives for negotiation.

## 3 A MarketSpace server

We have developed a MarketSpace server architecture and prototype based on the ideas presented above. The prototype was developed using SICStus Prolog and Prolog Objects [7].

The rôle of this server in the MarketSpace infrastructure is both to support human participants, with whom it will interact by emulating a web server, and programmed participants (agents), for which it will serve as a scheduler and communications mediator.

### The market server architecture

The MarketSpace server architecture has three main components (see Figure 4):

- the *kernel*, which handles the events that schedule activities in the system and communication with the outside world;
- the *protocol handlers*, which register the protocols they listen to and get the corresponding events from the event handler when data arrives;
- the *agent environment*, which implements the runtime environment for agents.

Two event types are built-in: The *time* events are used by system components (objects) that wish to

DEAL HEAD	
TIME	now until now+T
PARTICIPANTS	A=P1, B=P2
DEAL TYPE	"product trade..."
DEAL CONTENT	
ROLES	Seller(P1),Buyer(P2)
TRANSACTION	T1(P1->P2:X, P2->P1:Y)
ITEMS DESCRIPTION	
X: book{Title = "...", ...}	
Y: money{Currency = SKr, Amount = 100}	
TRANSACTION INFORMATION	
T1.delivery(X) = "mail"	
T1.payment(X) = "cash on delivery (COD)"	
T1.warranty(X) = none	
T1.delivery(Y) = "postal giro"	

Figure 1: Example of a possible EOI

be scheduled at regular time intervals and the *stream* events signal the arrival of data from the outside world to system components such as the protocol handlers.

### The object model

The implementation relies heavily on the Prolog Objects extension of SICStus Prolog. The basic objects (prototypes) are *root*, which adds initialization and finalization capabilities, and *persistent*, which also adds the ability to save (checkpoint) and recreate state from secondary storage. The persistence mechanism is also used for migration of objects.

### The kernel

The kernel is the main engine of the system and has two important rôles: one is to generate stream and time events, the other is to distribute events among subscribing objects. The kernel consists of the following components:

- the *stream handler* object, which handles the communication from the outside world with the system using socket and stream communication;
- the *time handler* object, which generates time and delay events;
- the *event handler*, which is the main kernel object, and handles scheduling of events to the system objects.

The kernel also gives higher level components the possibility to: subscribe for events, add events, and register new event types and protocols.

### The protocol handlers

The protocol handlers handle data arriving from the outside world. They register the protocols and subscribes to stream events from the event handler.

The *HTTP handler* provides World Wide Web server capabilities and forwards parsed HTTP messages to other objects. (A Prolog DCG parser could

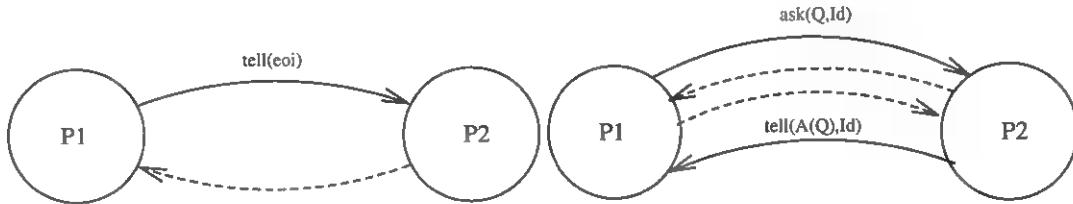


Figure 2: Tell and Ask

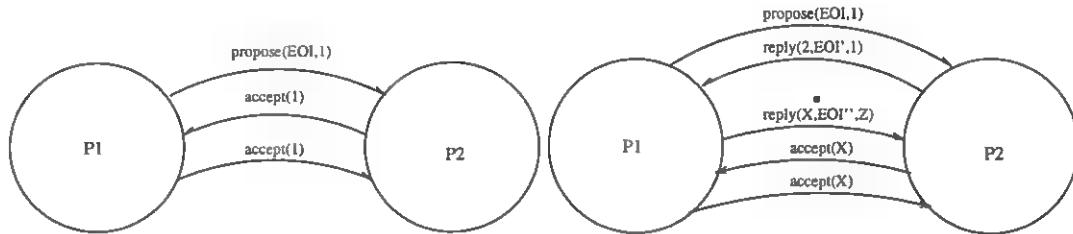


Figure 3: Negotiation

easily be derived from the HTTP specification [8].) Objects can subscribe to a specific path whereby web accesses matching that path will be forwarded to the subscriber. The following system components subscribe to HTTP events:

- the *html server*, which subscribes to the empty path and receives all web requests not caught by other subscribers, replying with files matching the web request;
- the *cgi-bin server*, which subscribes to the path "*cgi-bin*" and handles user registration and authentication;
- the *cgi object*, which subscribes to the path "*cgi-object*" and provides a way to inspect any runtime objects from the web.

The *DOP handler* handles communication with the DOP, Distributed Object Protocol, which is a simple protocol (defined by us) for distributing Prolog Objects.

The *TERM handler* handles communication with the TERM protocol. TERM messages are parsed into code and executed within the server context. This makes it possible to examine and control a running server which is very useful for debugging.

### The agent environment

At the top level of the architecture we have built a simple agent environment in which programmed participants in MarketSpace can reside. The agent environment implements a set of features useful for agents. These are:

- registration of requirements and description;
- agent communication;
- retrieval of information about other agents.

Although this environment offers some of the services needed by agents, it has to be extended. For example, there will be a need to manage ontologies for the content language(s) of agent communications.

Web-based users register in the server and are given a participant name in MarketSpace in the same name space, and indistinguishable from, the names of agents. Agents and humans need not, and should not, know if they are talking to humans or agents.

### Interests and interactions

For the present, the interests have a simple representation.

- Items, like products to sell, are represented as objects with a set of methods for accessing information about concepts and instance values.
- Deals are represented as different objects according to deal type.
- EOIs, Expressions Of Interest, are represented as objects with a list of deal objects. Methods for accessing EOI information (like deals) exists as well as set operations like *union*, *intersect* and *partition*

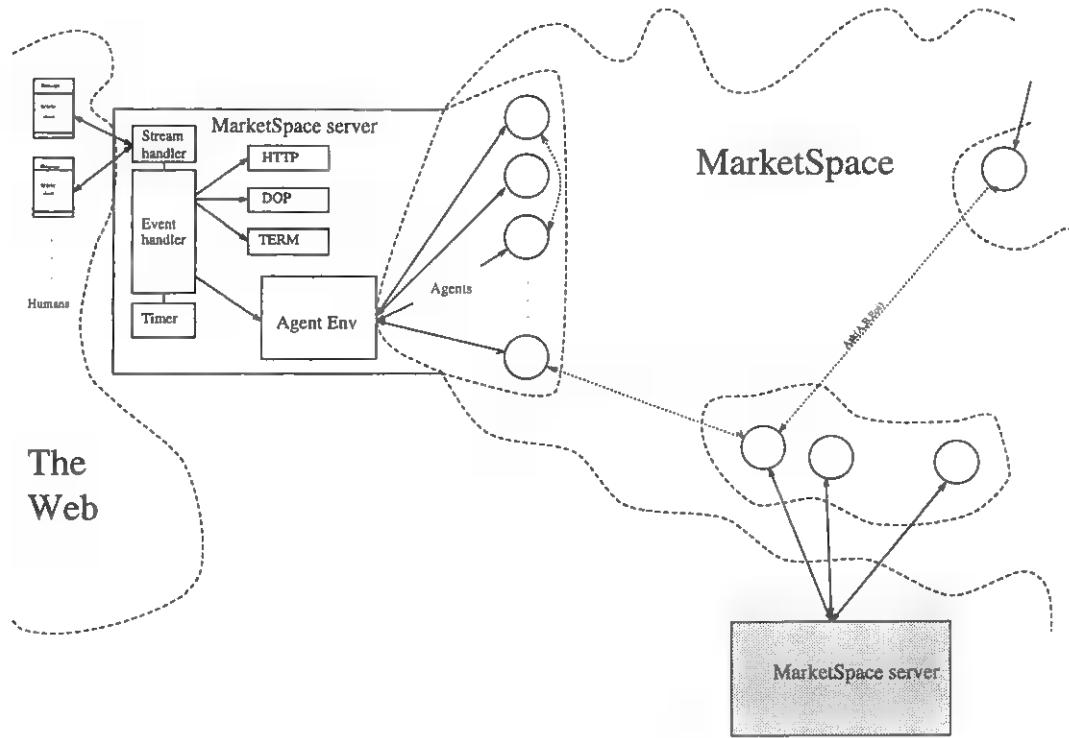


Figure 4: The basic MarketSpace architecture

The interaction model is implemented using interaction (or message) objects. They inherit from the persistent object (enabling migration) and have methods for accessing message information and sending messages to other agents.

### Example

Finally, we illustrate the use of this architecture and system by an interaction between two servers: the *blue server* and the *red server*.

The blue server provides a web interface to interact with human participants, who can create, change and remove interests from their interest store, and engage in negotiations, all through dynamic web pages. The blue server hosts a collector agent, which looks for new interests and forwards them to a broker agent at the red server.

The red server lets human participants specify their interests as URLs to web pages in which the interest is given in a special syntax. The red server fetches the specified web page and parses the interest. The red server also hosts a broker agent to which a human participant can delegate the task of negotiating a deal, giving upper and lower price limits.

In addition, a *log server* subscribes to log events from the other servers, and displays what is going on in MarketSpace.

### Example scenario

The following simple scenario illustrates a possible interaction in the marketspace.

- User *A* on the blue server specifies that she wants to buy a book. The user does this by creating an interest and specifying the rôle to act (buyer, seller) and the product to buy or sell. This interest is stored in a user store which is accessible by all agents in the marketspace.
- User *B* on the red server specifies that she wants to sell a book by specifying a URL to a web page (using the special format for interests). The web page is fetched and information about the interest is extracted and put in the user store.
- User *B* on the red server delegates her interest to a broker agent through an agent interface to which the broker agent has registered. When delegating the interest, the user specifies an acceptable price and a suitable starting price. During negotiation, the broker agent starts with the starting price and slowly goes to the acceptable price trying to get the best price.
- A collector agent in the blue server collects all local interests and sends them to the broker agent. At the moment this is done at regular

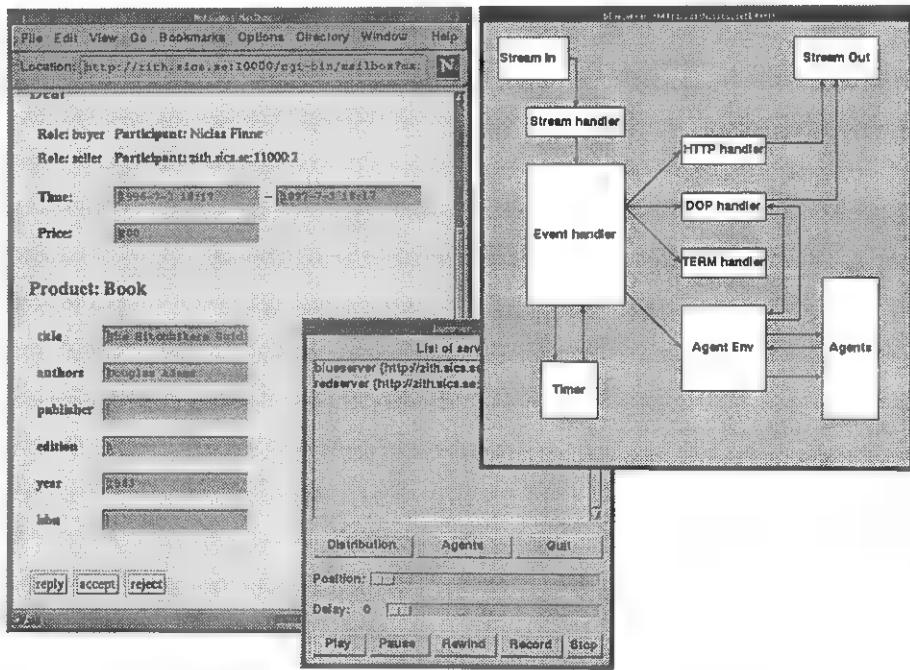


Figure 5: Snapshot of a proposal given to a user (left window) and the log handler

intervals, but in the future it will probably be some form of subscription. To collect all local interests, the collector agent starts by asking the administration agent for all local agents (which also includes users). Then the agent asks each such agent for their interests and when the agent happens to be a user, the user handler catches the message and answers it by fetching the users interests from the user store. The union of all interests found are sent to the broker agent at the red server.

- The broker agent receives the interest and checks for a matching interest by taking the intersection between the received interest and the delegated interest. If the intersection not is empty, the broker agent changes some information as for example price and sends a proposal to the agent that owns the interest. In this example it is user *A* at the blue server but it could just as well been a programmed participant.
- User *A* receives a message from the broker (see Figure 5) and the message is converted into a web page when the user reads it. The web page describes who the sender is, the type of message and its contents together with buttons to do various actions with it. For example, the user can choose between *accept*, *reject* and *reply* when replying to a proposal. The user chooses to negotiate by changing the price and select-

ing *reply*. A reply message is created and sent to the broker agent.

- The user and the broker agent negotiates until both accepts a deal (or someone rejects in which case the broker agent tries to find another agent to negotiate with).
- When the broker agent finally settles a deal, the user *B* is informed.

## 4 Conclusions and future work

We have defined a first version of the MarketSpace architecture for open agent-based market infrastructures, allowing human and programmed participants to co-exist smoothly. A first prototype has demonstrated the feasibility of the approach. This work is the starting point for a range of activities. In the short term future we will:

- investigate representation languages and ontologies for interests, involving issues such as product/service categories and transaction types;
- explore the integration of emerging payment mechanisms;
- complement the present platform with distributed naming and directory services, possibly based on KQML-like knowledge sharing;

- explore the possibility to support MarketSpace participants by “defection detection” and protocols for “social control” [9];
- develop a range of MarketSpace components to support various rôles, in particular professional vendors and suppliers of services, brokers and other mediation services, and interfaces to other market places, in particular services on the web, publishing their information and in effect creating a unified market.

Finally, for those who are interested in exploring systems of this kind using Prolog, we offer our present prototype as a starting point (see <http://www.sics.se/isl/commerce/>, where also other papers and more information on the project can be found).

[9] L. Rasmusson, A. Rasmusson, and S. Janson. Reactive security and social control. In *Proceedings of 19th National Information Systems Security Conference*. NCSC, 1996.

## References

- [1] Internet Shopping Network. URL: <http://www.internet.net>.
- [2] Polycon AB. The webra marketplace. URL: <http://www.polycon.fi/webra>.
- [3] Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of PAAM'96*, 1996.
- [4] Maksim B. Tsvetovatyy and Maria Gini. Toward a virtual marketplace: Architectures and strategies. In *Proceedings of PAAM96*, 1996.
- [5] Jim R. Oliver. On automated negotiation and electronic commerce. In *Proceedings of 29th Hawaii International Conference on System Sciences*. IEEE Computer Society Press, 1996.
- [6] Tim Finin and Richard Fritzson. KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, 1994.
- [7] Swedish Institute of Computer Science. *SICStus Prolog User's Manual*, 1995. Also available via <http://www.sics.se/isl/sicstus.html>.
- [8] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol - HTTP/1.0. Available at <http://www.w3.org/pub/WWW/Protocols/HTTP1.0/draft-ietf-http-spec.html>.



# A Peer-to-Peer Software Metering System

Bruce Schneier      John Kelsey

{schneier,kelsey}@counterpane.com

Counterpane Systems, 101 East Minnehaha Parkway, Minneapolis, MN 55419

## Abstract

We present two software-network payment systems, designed so that every user is capable of both buying and selling. One system uses online clearing; the other uses offline clearing.

## 1 Introduction

Internet commerce requires new payment methods, and several have already been developed [OO90, OO92, FY92, Bra93a, Bra93b, MN93, CR93, Fer94, LMP94, Oka95, MN95, BGH+95, ST95, ?]. This paper describes a specific kind of payment system in which each user can transfer money to each other user: a peer-to-peer payment system. This can be thought of as a credit card account or a checking account. At regular intervals, these electronic payments are translated into real-world payments. Although this step is important to the users of these protocols, it's not a cryptographic operation at all; just a matter of executing EFTs or writing and mailing out checks.

This paper is organized as follows. Section 2 is a discussion of the objectives and design principles for this kind of system. In Section 3 we describe two example systems, one using online clearing, the other, offline clearing with public key signatures. Neither of these is meant as a final system, but both serve as good outlines to build a working system around.

## 2 Objectives, Design Principles, and Options

The ultimate objective is to have a payment system as handy for doing internet trade as cash, checks, or credit cards are for doing day-to-day trade. Whatever this is, it must meet the following criteria:

1. *Secure.* The system must be secure in the sense that it must not be possible to convince someone they have received money when they have not. It must also not be possible to forge a payment from someone else, or to replay payments. Finally, it mustn't be possible to violate other design specifications of the system, for example by tracing other peoples' payments.
2. *Cheap.* The system shouldn't require the user to purchase expensive equipment (other than a PC). The operating costs of the system should be low enough that nobody has to pay a large fixed monthly fee. Indeed, a small fixed monthly fee may be all anyone pays, if operation is cheap enough. In a computational sense, nobody should have to do too many complicated computations to pay someone a dime.
3. *Widely Available.* For maximum availability, the client (peer) software should be partially or completely available in source-code—participation in the payment system might be the paid for monthly, along with settling of bills as needed.
4. *Peer-to-Peer.* Each user should be able to either send or receive payments without complicated registration procedures.

There are several tradeoffs. For example, making a payment protocol computationally light usually means using few or no public key operations. However, this seems to require some kind of interaction with a central server, which drives up communications requirements and operating costs.

In this paper, we attempt to stay within the constraints of what is implementable in software on standard computers of today, using a good cryptographic function library. Individual users (peers) may have secrets, but these secrets have relevance only to those users. Only the bank is allowed to hold any global secrets.

## 2.1 Notation

Notation in the remainder of this document is as follows:

- $Enc_{K_A}(X)$  means that  $X$  is encrypted under key  $K_A$ , using some symmetric encryption algorithm.
- $Auth_{K_A}(X)$  means that  $X$  is authenticated under key  $K_A$ , using some symmetric message authentication algorithm.
- $EncAuth_{K_A}(X)$  means that  $X$  is authenticated and encrypted, using symmetric algorithms, under keys derived from  $K_A$ .
- $PKEnc_{K_A}(X)$  means that  $X$  is encrypted under public key  $K_A$ .
- $Sign_{K_A}(X)$  means that  $X$  is digitally signed under private key  $K_A$ .
- $X, Y$  means that  $X$  is concatenated with  $Y$ .

## 3 Online Clearing Systems

Online clearing systems are simpler than offline systems, at least in principle, since both users can authenticate themselves to the Bank's trusted server instead of to each other. There are no certificates to worry about, and the most important security operations take place on a secured machine. Also, this system obviates the need for computationally expensive public-key operations (and the associated licensing issues). The cost of this is that the Bank has to maintain a constant online presence, and can easily become the bottleneck of the system.

In this section, we briefly discuss one example of an online system. In this system, we require the person accepting the payment to have a reasonably accurate clock, and to keep some short-term memory about transfers that have been accepted recently, to prevent trivial replay attacks. When Alice wants to transfer money to Carol, she first gets an electronic note of approval from the Bank for this transfer, which includes an expiration time (after which Carol should not accept it). Then, she sends this note to Carol to convince her that the transfer has indeed taken place. This turns out to be very close to the Kerberos protocol [Sch96].

## 3.1 Variables

1.  $S_A$  and  $S_C$  are Alice and Carol's sequence numbers. These must never repeat or go backwards—instead, they increment one value at a time. Suspicious jumping around by the sequence number must always be noted by the bank, and should initiate corrective action of some kind.
2.  $ID_A$  and  $ID_C$  are Alice and Carol's unique 64-bit ID numbers.
3.  $K_A$  and  $K_C$  are the keys shared between Alice and the Bank, and Carol and the Bank, respectively. The keys are derived based on a master key held by the bank,  $K_*$ , according to the relation
$$K_i = E_{K_*}(ID_i)$$
4. The audit-log is the log of all previous payments, kept by Alice's software. By including this hash, Alice commits to the current entries in the log; if she alters these later, it will be detected by the Bank [SK96].
5. A Timestamp is a 32-bit representation of a given time and date.

## 3.2 Basic Protocol: Alice Transfers Money to Carol

1. Alice forms

$$\begin{aligned}U_0 &= \text{"Request for Transfer Authorization"} \\S_A &= \text{Current sequence number for Alice-incremented immediately} \\V &= \text{hash(Audit Log)} \\W &= \text{Amount of Transfer} \\X_0 &= U_0, S_A, V, W, ID_C \\K_0 &= \text{a random message key}\end{aligned}$$

and sends to the Bank

$$M_0 = ID_A, EncAuth_{K_A}(K_0), EncAuth_{K_0}(X_0)$$

2. The Bank receives and decrypts this. If the message doesn't decrypt or authenticate properly, the Bank responds with a simple error message. If the sequence number or hash of the audit log is wrong, then it must begin corrective action. This will be discussed further below. If the amount of the transfer is more than the

amount in Alice's account, the ID for Carol is invalid, or anything else is wrong with the message, then a simple error message is sent to Alice. However, if nothing is wrong with  $M_0$ , then the Bank forms

$$\begin{aligned} U_{1a} &= \text{"Transfer Authorization"} \\ T_E &= \text{Expiration time of authorization} \\ Y_1 &= U_{1a}, \text{hash}(M_0)ID_A, ID_C, W, T_E \\ K_1 &= \text{a random message key} \\ Z_1 &= \text{EncAuth}_{K_C}(K_1), \text{EncAuth}_{K_1}(Y_1) \\ U_{1b} &= \text{"Transfer Verification"} \\ S_C &= \text{Carol's current sequence number."} \\ ID_A, ID_C, W & \text{from above step.} \\ X_1 &= U_{1b}, \text{hash}(M_0), ID_A, ID_C, W, S_C, \\ T_E & \\ K_2 &= \text{a random message key} \end{aligned}$$

and sends back to Alice

$$\begin{aligned} M_1 &= \text{EncAuth}_{K_A}(K_2), \\ &\text{EncAuth}_{K_2}(X_1, Z_1) \end{aligned}$$

3. Alice receives this, and verifies everything she can verify. If there are no problems, she forms

$$\begin{aligned} U_2 &= \text{"Payment"} \\ Z_1 & \text{from above.} \end{aligned}$$

and sends to Carol

$$M_2 = U_2, Z_1$$

At this point, Alice updates her audit log to include this transaction. This is done by appending  $ID_C, W$  to the previous contents of the audit log. In the event of any trouble verifying that this payment arrived, Alice and/or Carol contact the Bank.

4. Carol appends  $ID_A, W$  to her audit log, and adjusts her internally-carried balance by the amount of the transfer.

### 3.2.1 Security of the Protocol

#### 1. Replays

- (a) The Bank would recognize any replay attempts immediately because of Alice's sequence number. Only an attacker that could alter or forge messages from Alice could replay a message with a new sequence number, without being caught at it.

(b) Alice would recognize any replay attempts because the hash of her previous message is embedded inside the second protocol message.

(c) Carol would recognize replay attempts directed at her by the replayed timestamp and incorrect sequence number. Note that there needs to be some room for error in both these values for Carol, because messages may not always arrive in the order they were sent, and because most computer clocks aren't highly accurate.

(d) It's not possible to alter individual parts of transmitted messages because of the authentication being used. It's not possible to alter individual messages in a protocol because they typically contain the hash of the prior message.

2. *Forgeries* Forgeries should be very difficult to carry out, if the symmetric authentication scheme is acceptably strong.
3. *Information Leaks* Because of the use of a new message key for each message, it should be very hard for any information to leak. Within the message formats, only the ID of the person transferring money to someone else is leaked in any message.
4. *Other Attacks* There is a trivial attack in which Carol simply discards a payment to her by Alice, which means that Alice and the Bank will eventually need to resolve this. This is fundamentally a nuisance attack.

### 3.3 Secondary Protocol: Alice Reconciles

Occasionally, communications failures, implementation errors, or fraudulent transfers may leave Alice and the Bank unsynchronized. The solution is to go through a more complicated protocol to verify that all is well. This is also done from time to time to verify that Alice has received all the deposits that she should have received by now. During the ordinary transfer protocol, the Bank can require a reconciliation protocol before it will allow Alice to carry out any transfers, or Carol to receive any. This protocol is always required before depositing or withdrawing any money from the bank.

1. Alice forms

$U_0$  = "Request for Reconciliation"  
 $R_0$  = a random challenge  
 $S_A$  = Alice's current internal sequence number  
 $X_0 = U_0, S_A$   
 $K_0$  = A random message key

she then sends to the Bank

$M_0 = ID_A, EncAuth_{K_A}(K_0),$   
 $EncAuth_{K_0}(X_0).$

2. The Bank verifies the authentication, and then forms

$U_1$  = "Reconciliation Challenge"  
 $S_A^*$  = Bank's current idea about what  $S_A$  should be.  
 $K_1$  = A random message key  
 $X_1 = U_1, \text{hash}(M_0), S_A^*$

it then sends to Alice

$M_1 = EncAuth_{K_A}(K_1), EncAuth_{K_1}(X_1).$

3. Alice verifies the authentication, and that the hash of the previous message is included. She then forms

$U_2$  = "Reconciliation Response"  
 $X_2 = U_2, \text{hash}(M_1)$ , Full Authentication Log  
 $K_2$  = a random message key  
 $S_A = \max(S_A, S_A^*) + 1$

she then sends to the Bank

$M_2 = EncAuth_{K_A}(K_2), EncAuth_{K_2}(X_2).$

4. The Bank, after verifying all this information, either is or is not willing to let Alice start transferring money again. Thus, it forms

$U_{3a}$  = "Reconciliation Success Message"  
 $U_{3b}$  = "Reconciliation Failure Message"  
 $K_3$  = a random message key  
 $X_{3a} = U_{3a}, \text{hash}(M_2), S_A$ , Account Balance  
 $X_{3b} = U_{3b}, \text{hash}(M_2)$ , Problem Description

and sends to Alice, depending on the circumstances, one of the following:

$M_{3a}$	=	$EncAuth_{K_A}(K_3),$
$EncAuth_{K_3}(X_{3a}).$		
$M_{3b}$	=	$EncAuth_{K_A}(K_3),$
$EncAuth_{K_3}(X_{3b}).$		

At the end of this protocol, Alice and the Bank should have some common new sequence number, and either they should agree on what has happened, or there will be some kind of investigation going on, and Alice should know the basic kind of problem and what to do next.

### 3.4 What Can Go Wrong?

The most dire failure would involve a security breach at the Bank. Maintaining an authenticated, physically secure and separate log would be one way to ensure that a bank could recover from this kind of failure. Note that each payment from a user contains a current hash of her log.

The user's PC can lose security, in almost the same sense that someone can lose a credit card or checkbook. This should be rare, but it will generally have to be dealt with by humans. The software maintains a log, authenticated by chained hashes. Each payment commits to the current value of the log; that is, the hash of the log up until now.

The authentication and/or encryption functions can be broken. If this happens, the system must be recalled and fixed, and lots of money will be lost. This gives us a lot of incentive to choose our authentication and encryption functions well. For example, Triple-DES [NBS77] or Blowfish [Sch94] in CBC-mode might be used to encrypt, and a keyed hash might be made from SHA1 [NIST93], using a well-thought-out construction such as [PvO95].

## 4 An Off-line Payment System

The offline system can be imagined as a checking account: users are allowed to write checks for whatever's in their accounts. The software will not allow them to overdraw, but it is assumed that fraudulent users can change their software to allow this. We assume that we know how to deal with people who don't pay their bills: they can overdraw for a limited span of time, and the bank's collections department will wind up trying to get the money back from them. The offline system uses some public key operations instead of interactions with the

Bank. Although certificates are used, there is no CRL. Instead, certificates have a very short lifetime, perhaps a week. After that time, the user must connect to the bank to get a new certificate.

There are two routine operations: Payment and Reconciliation. In payment, one user (Alice) will pay \$X to another user (Carol). In reconciliation, a user (Alice) will interact with the Bank to allow her to continue using the system. The system assumes that both Alice and Carol have reasonably accurate clocks.

## 4.1 Payment

In this protocol, Alice makes a payment to Carol by sending Carol a “draft.” That is, a timestamped, digitally signed authorization for the bank to move \$X from Alice’s account into Carol’s. Each draft has a unique identifying number, which prevents the bank from ever accepting replays, and gives Alice a way to refer to disputed payments.

Variables:

1. Timestamp is the current timestamp, in some standard format. The timestamp must never repeat, so it should include everything from centuries to seconds. An example might be “19951225010000”.
2. The audit-log is the log of all previous payments. By including this, Alice commits to the current contents of her log each time she makes a payment. Later changes are detected by the Bank during reconciliation.
3. The Draft is the order specifying a draft sequence number, the account number of the payor, and the account number of the payee.
4.  $Cert_A$  is Alice’s certificate, attesting to the current valid linkage between her public signing key and her ability to write drafts on a given account id.
5.  $PK_A$  is Alice’s public signing key.

This is the protocol by which Alice pays Carol:

1. Alice forms

$$U_0 = \text{“Check Transmission”}$$

$$T_0 = \text{timestamp}$$

$$V_0 = \text{hash}(audit - log)$$

$$X_0 = U_0, V_0, T_0, \text{Draft}$$

$$S_0 = \text{Sign}_{SK_A}(X_0)$$

$$K_0 = \text{a random message key}$$

and sends to Carol

$$\begin{aligned} M_1 &= Cert_A, & PKE_{PK_C}(K_0), \\ &Enc_{K_0}(X_0, S_0) \end{aligned}$$

2. Carol verifies Alice’s certificate and timestamp. She then verifies the signature on Alice’s draft. If all is well, she knows that the payment has occurred.

This protocol is somewhat computationally expensive. Alice must perform a signature, and Carol must perform two verifications (though she may want to maintain lists of valid certificates, so she doesn’t perform the certificate verification more often than necessary). Note that all payments from Alice to Carol appear in the clear; if a secure connection has already been made, then this will work well. If not, it may be necessary in some cases to establish a secure connection; this implies more public key operations for a strong key exchange. Also note that nothing keeps Carol from refusing to acknowledge Alice’s payment; Alice can contact the bank and send a signed request to have a stop-payment put on that draft.

## 4.2 Deposit

Deposit simply consists of Carol sending to the Bank (perhaps by e-mail) the digitally signed draft from Alice. Since the draft names Carol, there is no risk of redirection. An active attack can prevent the bank from receiving the deposit. The Bank returns a receipt, also digitally signed. Because each draft must be different, there is no risk of replay attack, if things are done properly.

This is the protocol by which Carol deposits a draft:

1. Carol forms

$$U_1 = \text{“Deposit Request”}$$

$$T = \text{Timestamp}$$

$$V_1 = X_0, S_0 \text{ from the previous protocol}$$

$$X_1 = U_1, T, V_1.$$

$$S_1 = \text{Sign}_{SK_C}(X_1)$$

$$K_1 = \text{a random message key}$$

and sends to the Bank

$$M_1 = PKE_{PK_C}(K_1), Enc_{K_1}(X_1, S_1)$$

2. The Bank verifies that all is well. If so, it forms

$$U_2 = \text{"Deposit Receipt"}$$

$$R = \text{Receipt}$$

$$X_2 = U_2, R, \text{hash}(M_1)$$

$$S_2 = \text{Sign}_{SK_B}(X_2)$$

$$K_2 = \text{a random message key}$$

and sends to Carol

$$M_2 = PKE_{PK_C}(K_2), Enc_{K_2}(X_2, S_2)$$

If Carol never receives  $M_1$ , she restarts the protocol later, or eventually notes it when she reconciles.

### 4.3 Reconciliation

Reconciliation occurs when Alice connects with the Bank to send in her logs, including copies of all drafts she has written, and to receive a new certificate. User certificates expire often.

1. The Bank forms

$$U_3 = \text{"Reconciliation Request"}$$

$$R_3 = \text{a random 64-bit value}$$

$$X_3 = U_3, R_3$$

$$S_3 = \text{Sign}_{SK_B}(X_3)$$

$$K_3 = \text{a random message key}$$

and sends to Alice

$$M_3 = PKE_{PK_A}(K_3), Enc_{K_3}(X_3, S_3)$$

2. Alice forms

$$R_4 = \text{a random 64-bit value}$$

$$U_4 = \text{"Reconciliation Response"}$$

$L$  = the log of payments and deposits since last reconciliation, noting any for which she received no receipt, and any for which she wishes to dispute payment.

$$X_4 = U_4, \text{hash}(M_3), R_4, L$$

$$S_4 = \text{Sign}_{SK_A}(X_4)$$

$$K_4 = \text{a random message key}$$

and sends to the Bank

$$M_4 = PKE_{PK_B}(K_4), Enc_{K_4}(X_4, S_4).$$

3. The Bank verifies that all is well. If so, it responds by forming

$$U_5 = \text{"Reconciliation Receipt"}$$

$$C_A = \text{New Certificate for Alice with a new timestamp}$$

$$X_5 = U_5, \text{hash}(M_4), C_A, \text{Ending Balance}, \text{Timestamp}$$

$$S_5 = \text{Sign}_{SK_B}(X_5)$$

$$K_5 = \text{a random message key}$$

and sends to Alice

$$M_5 = PKE_{PK_A}(K_5), E_{K_5}(X_5, S_5)$$

If there are disputed payments or other problems, there will be a request for more information sent. This should lead to an online session, telephone call, or face-to-face discussion of the user's complaint. Dispute resolution falls outside of the scope of the payment system.

### 4.4 Error Recovery, Fraud Detection, and Auditing

A continual audit trail is kept in the user's PC, and its current value is committed to each time a payment is made. Forged transactions will be caught during reconciliation, as will almost all other problems. People who overdraw their accounts, exceed their credit lines, or fail to pay their bills have either had a software flaw, or hacked their software to do these things. They will not be issued a new certificate until they resolve the problem. This limits damage from individual security failures to a small period of time, perhaps no more than a week.

### 4.5 What Can Go Wrong?

Compromise of the private signing key used to create certificates is probably the most catastrophic thing that could happen. This must be guarded against strongly, perhaps maintaining the key in a tamper-resistant token under good physical security. Compromise of this key would allow an attacker to write an endless stream of bad checks on anyone's account.

The user's PC can lose security, in almost the same sense that someone can lose a credit card or checkbook. This should be rare, but it will generally have

to be dealt with by humans. The software maintains a log, authenticated by chained hashes. Each payment commits to the current value of the log: i.e., the hash of the log up until now. Since certificates expire often, this will be dealt with soon after it's discovered by freezing the account that has been compromised until the problem can be dealt with. In this system, another thing that can happen is that a user's clock can be fouled up, to trick him into accepting a stale certificate. Such attacks must be guarded against.

The authentication, signing, and/or encryption functions can be broken. If this happens, the system must be recalled and fixed, and lots of money will be lost. This gives us a lot of incentive to choose our functions well. For example, we might use 1280-bit RSA [RSA78] SHA1 [NIST93] for our signatures.

## 4.6 Additional Comments and Extensions

Note that it is possible to give the payor anonymity from the payee. This is done by giving each user a large number of IDs, each with a different public signing key and certificate. This gives no anonymity from the bank.

## 5 Retrofitting Existing Systems

There are several reasonably good methods of paying for things available now. Small variations in these could be adapted to this kind of an application. The advantages of using an already fielded system are that there is no need to build the system's infrastructure, and that a fielded system may already have had many of its flaws worked out. Among the possible disadvantages are that many currently-fielded payments systems don't meet the specific needs of this application, and that there are often hardware requirements and licensing fees.

- DigiCash is payer-anonymous electronic cash. It is currently available. The cost per transaction appears to be too high, and there are some practical security problems with any payer-anonymous system that need to be worked out, but this kind of system might be worth using for this application at some point.

- Some schemes simply transfer users' credit card numbers around. These have obvious security problems involving replay and forgery attacks. However, one way to implement any of the systems discussed above would be for the bank to bill users' credit card for any money owed. To avoid high transaction costs, the bank would bundle many small transactions per month before billing them—perhaps only in \$50 increments—from the user's card. The advantage of this would be in low start-up costs. Ideally, the system would also be able to credit money to their credit cards, or transfer it into their bank accounts. This would simplify the mechanics of running this kind of system.
- There are micropayment schemes, such as Millilient, Payword, and Micromint, that allow for small monetary values to be transferred among parties. However, micropayment schemes generally have to solve somewhat different problems than a peer-to-peer metering system, so they make somewhat different tradeoffs.

## 6 Conclusion

The future of the Internet is one where many people are both producers and consumers of information. Any payment system needs to reflect this. This work, while no means complete, shows the kind of directions necessary for peer-to-peer payment systems.

## References

[BGH+95] M. Bellare, J.A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner, "iKP - A Family of Secure Electronic Payment Protocols", *The First USENIX Workshop on Electronic Commerce*, USENIX Association, 1995, pp. 89-106.

[Bra93a] S. Brands, "Untraceable Off-line Cash in Wallets with Observers," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994 pp. 302-318.

[Bra93b] S. Brands, "An Efficient Off-line Electronic Cash Systems Based on

the Representation Problem," C.W.I. Technical Report CS-T9323, 1993.

[CR93] CitiBank and S. S. Rosen, "Electronic-Monetary System," International Publication Number WO 93/10503; May 27 1993.

[Fer94] N. Ferguson, "Extensions of Single-Term Coins," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 292–301.

[FY92] M. Franklin and M. Yung, "Towards Provably Secure Efficient Electronic Cash," Columbia Univ. Dept of C.S. TR CUCS-018-92, April 24, 1992. (Also in Icalp-93, July 93, Lund Sweden, LNCS Springer-Verlag).

[LMP94] S. H. Low, N. F. Maxemchuk and S. Paul, "Anonymous Credit Cards," *The Second ACM Conference on Computer and Communications Security*, ACM Press, 1994, pp. 108–117.

[MN93] G. Medvinsky and B. C. Neuman, "Netcash: A Design for Practical Electronic Currency on the Internet," *The First ACM Conference on Computer and Communications Security*, ACM Press, 1993, pp. 102–106.

[NBS77] National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard," National Bureau of Standards, U.S. Department of Commerce, Jan 1977.

[MN95] B. C. Neuman and G. Medvinsky, "Requirements for Network Payment: The NetCheque<sup>TM</sup> Perspective," Compcon '95, pp. 32–36

[NIST93] National Institute of Standards and Technology, NIST FIPS PUB 180, "Secure Hash Standard," U.S. Department of Commerce, May 93.

[Oka95] T Okamoto, "An Efficient Divisible Electronic Cash Scheme," *Advances in Cryptology—CRYPTO '95 Proceedings*, Springer-Verlag, 1995, pp. 438–451.

[OO90] T. Okamoto and K. Ohta, "Disposable Zero-Knowledge Authentication and Their Applications to Untraceable Electronic Cash," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 481–496

[OO92] T. Okamoto and K. Ohta, "Universal Electronic Cash," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 324–337.

[PvO95] Preneel and P.C. van Oorschot, "MDX-MAC and Building Fast MACs from Hash Functions," *Advances in Cryptology—CRYPTO '95 Proceedings*, Springer-Verlag, 1995, pp. 1–14.

[RSA78] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, v. 21, n. 2, Feb 1978, pp. 120–126.

[Sch94] B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 191–204.

[Sch96] B. Schneier, *Applied Cryptography, Second Edition*, John Wiley & Sons, 1996.

[SK96] B. Schneier, J. Kelsey, "Automatic Event-Stream Notarization Using Digital Signatures," in *Advances in Cryptology, Proceedings of the Cambridge Protocols Workshop 96*, Springer-Verlag, 1996, pp. in preparation.

[ST95] M. Sirbu and J. D. Tygar, "NetBill: An Internet Commerce System Optimized for Network Delivered Services," Compcon '95, pp. 20–25.

[TSMW95] J. M. Tenenbaum, C. Medich, A. M. Schiffman, and W. T. Wong, "CommerceNet: Spontaneous Electronic Commerce on the Internet," Compcon '95, pp. 38–43

## Modeling the Risks and Costs of Digitally Signed Certificates in Electronic Commerce

Ian Simpson  
is2a+@cmu.edu

Carnegie Mellon University  
Pittsburgh, PA 15213-3890

### Introduction

Public key certificate infrastructures are being rapidly deployed to support such diverse applications as secure electronic mail, code authentication, and electronic commerce. Because of the paucity of practical experience operating large scale public key certificate infrastructures, many technical design decisions are being made based on untested assumptions about patterns of certificate use and the costs and benefits of various design choices.

The present paper presents a quantitative model of the financial costs and risks of operating a public key certificate system for electronic commerce. A better understanding of the ways in which the technical components will work together in various commercial contexts may help in the design of systems and standards that suit the needs of users, and that stand the test of time. The bearers, issuers, and recipients of certificates can benefit from a better understanding of how operational decisions will affect their individual and collective costs and risks.

As with any model building activity, the goals are several. First, we hope to identify those parameters which most influence the performance of a public key infrastructure system. This will guide developers to focus their attention on the key variables. Second, we hope to provide some intuition with respect to the tradeoffs in operating cost and risk that result from various system design choices. Finally, a model should permit rapid incorporation of empirical data that will emerge from leading edge systems so that a refined model can guide future operational decisions.

### General Approach

Examining certificates in the context of electronic commerce imposes constraints that narrow the investigation to a manageable scope: First, there is a monetary measure of loss, gain and risk. Second, the potential risks to the participants are bounded and measurable. The mechanisms of electronic commerce constrain both the expected loss, and the worst-case losses that are possible. Third, prior business practice and commercial law govern conventional commerce, and they serve as indicators of what capabilities may be considered desirable. This research further narrows its

focus to the technical parameters for the design and operation of certificate systems. The choice of *certificate lifetime*, the period of time during which a certificate is considered valid, directly affects the level of reliability and security that can be achieved, as well as affecting the cost of implementing and maintaining the certificate system.

A probabilistic model of the certificate process was created that examines the effects of the design and operational parameters on the benefits, costs, and risks of the parties involved, focusing specifically on the effect of certificate lifetimes. The model identifies the main elements of the certificate process and the first-order relationships between them. It includes a number of design and operational parameters which can be directly and deterministically chosen by the affected parties, such as the certificate lifetime. It includes also a number of uncertain variables that define the environment within which the system functions, such as the rate of occurrence of various sorts of compromise, or the effectiveness of fraud detection measures. It was not possible to determine the precise values of all the variables; considerable uncertainty exists that cannot be substantially reduced until electronic commerce and certificates are, over a period of time, thoroughly examined in an operational setting. For this reason, the behavior of the model was examined over a range of potential values: Sensitivity analysis of the uncertain variables was used to determine the extent to which they influence the model outcome. Parametric analysis of the choice variables was conducted across ranges that are likely scenarios for electronic commerce. In cooperation, these allowed the identification of regions of the parameter space that demonstrated specific system behaviors of interest.

The preparation of this paper uncovered a paucity of quantitative studies of the risks and costs of certificates as used in electronic commerce. This paper attempts to model the risks and costs of a certificate system used for electronic commerce. It represents an early attempt at modeling such a system, constructed in the absence of detailed empirical data. As such, it is necessarily an abstraction, and is more useful for understanding the relationships of the various processes than for any real predictive power. Subsequent versions of this model, will benefit from the lessons learned from this effort, and from real-world data as it becomes available. The model focuses attention on which

variables are most sensitive and therefore need to be measured.

## Modeling Methodology

A probabilistic model was constructed, using Analytica decision support software. The model was constructed to represent the effects of certificate lifetimes on the cost and the risk of loss experienced by merchants, and on the cost to the certifying authority (CA) of issuing and maintaining certificates. Parametric was used to examine the effects across ranges of parameters and uncertain variables.

The model describes the following electronic commerce situation: A single certifying authority (CA) issues certificates to customers, re-issuing them periodically. The customers present these certificates to merchants to show that they are trustworthy. Trusted customers may transact with the merchants to purchase goods and services. The CA incurs costs in a number of ways: The process of issuing certificates, results in a cost that is proportional to the number issued. A large part of the issuance cost is the off-line validation of the information that is to be included in the certificate. The CA also may incur costs to periodically re-validate the certificate information. It also must maintain a certificate revocation list (CRL), which incurs a cost proportional to the volume and the type of accesses by merchants who wish to ensure the validity of certificates that have been presented to them. Merchants risk monetary loss by accepting certificates: a merchant who extends trust in error risks a loss (nonpayment) from the transaction. While certificates remain valid (neither expired nor revoked) they may be used repeatedly, so a single "bad certificate" can be used in multiple transactions in which merchants suffer a loss. Merchants also incur a cost in checking the CRL. A number of CRL options are available to them, each with differing cost and risk implications.

The lifetime of a certificate is the span of time during which it is considered valid. When a certificate expires, the CA issues a replacement to the customer. The CA must maintain computer systems and connectivity to adequately serve its customer base. Certificate lifetimes affect the risk of loss experienced by the merchant. Different types of certificate error may have differing chances of ending at the expiration of the certificate in question. So shortening the certificate lifetime may have differing effects on the relative risks resulting from different types of certificate compromises.

Certificate compromise has two causes. First, the certificate may be used by an unauthorized party. This implies that the mechanism for disseminating or using the certificate has been compromised, either accidentally or intentionally. This is termed *mechanism compromise*. "Accidentally" making

unauthorized use of another party's certificate for commercial transactions is very unlikely-- The model assumes that all mechanism compromise is a result of deliberate fraud. To the extent that mechanism compromise is tied to a specific private key, it is likely to end with the expiration of the certificate (assuming that new certificates imply new key pairs). A second cause of compromise is that the certificate itself may be inaccurate. For our purposes, certificates are assumed tamper-proof implying either that information the CA used to issue the certificate was in error, or that the information was accurate at the time of issuance, but has since become obsolete without the CA's knowledge. This is termed *information compromise*, and may occur either as a result of error or of deliberate fraud. Expiration of a certificate may or may not limit compromise if the CA's information is wrong: it depends on the CA's policy and effectiveness in updating the information associated with the certificate, in its own database.

Figure 1 is a simplified representation of the model used to examine the certificate system. The model assumes a stable and homogeneous merchant base, customer base, and transaction volumes. The CA's costs are determined by the number of certificates issued (which is determined by certificate lifetime), its information maintenance policies, and the number of CRL checks initiated by merchants. Merchant costs are determined by the number of CRL checks they initiate, which is influenced by their certificate validation policies. Merchant losses are determined by the rate and average duration of certificate compromises, which are in turn influenced by both the merchants certificate validation policies and the CA's information maintenance policies.

## Three Electronic Commerce Scenarios

All electronic commerce is not the same. The model examines low-end, mid-range, and high-end examples, and the differences between them.

**Microtransactions:** Low cost information goods, with an average daily spending of about a dollar.

**Credit Card model:** Mid-range information and physical goods, with an average daily spending of about 5 dollars.

**Corporate Purchasing:** Potentially numerous and expensive items, with an average daily spending that could exceed 100 dollars or even more

The optimal mix of operational choices that will minimize risks and costs is likely to differ among the above cases.

## Assumptions

A number of assumptions and abstractions were necessary to reduce the model to a tractable size and complexity.

- There is one CA, and many merchants.
- Customers are not explicitly modeled. They are included implicitly through merchant transaction rates.
- The CA is trusted by both customers and merchants.
- CAs and merchants always act in good faith, and their private keys are always secure.
- The cost to the CA of issuing certificates, validating certificate information, and maintaining a CRL is modeled.
- The risk to merchants of accepting certificates is modeled.
- The cost to merchants of checking the validity of certificates against the CRL is modeled.
- The rate of certificate compromise is assumed constant over time
- The rate of compromise detection is assumed constant over time

## Detailed Treatment

The following is a description of the mathematical underpinnings of the Credentials model, implemented in Analytica.

### Direct Costs to the CA and Merchants

The costs portion of the model is a deterministic simulation of the direct costs of participating in the certificate system.

The cost to the CA is the sum of the following:

- The cost of issuing the certificates
- The cost of information verification at subscription time
- The cost of subsequent information verification
- The cost of maintaining a certificate revocation list (CRL)

The cost to the Merchant consists of the following:

- The cost of accessing the CA's CRL

### Risks to the Merchants

The risks portion of the model is a probabilistic simulation of the financial risk to the merchants participating in the certificate system. Merchants who extend trust based on flawed premises incur a risk of nonpayment. Malefactors may continue to carry out transactions until the compromise is terminated. We assume merchants bear all the costs of fraud. These costs could be shifted -- but not reduced -- where a payment processor such as a credit card association assumes certain loss risks in return for an insurance premium in the form of a discounted payment to the merchant.

### Types of Compromise

*Compromise* of the certificate system allows someone to commit fraud.

This model distinguishes between two broad categories of compromise: *information compromise*, in which the information underlying the subscription is attacked, and *mechanism compromise*, in which the certificate system is attacked.

Information compromise at subscription time is termed *initial* information compromise. Information compromise after subscription time is termed *subsequent* information compromise. Mechanism compromise is assumed to always occur after subscription time. The probability density function of compromise as a function of time since certificate issuance for each type of compromise is assumed stationary.

### Delay Between Compromise Discovery and Compromise Termination

Information compromise is terminated when it is discovered by a merchant or by the CA, and knowledge of the compromise is propagated to the merchants' certificate revocation lists. There may be a time lag between

Information compromise that occurs at subscription time may be detected at time 0 by the CA, or may be detected later by either the CA or a merchant. If the compromise is detected at time 0, propagation of discovery is not needed because the certificate is never issued. Information compromise that is detected after issuance may involve a lag between discovery of the compromise and the termination of the compromise. Termination only occurs after propagation of discovery to the merchant community is accomplished. This lag may consist of two parts: the time before the knowledge is incorporated into the CA CRL (assumed

nil if the CA is the discoverer) and the time before the knowledge is generally available in merchant CRLs. These lag times depend on the merchants' information update policies.

Termination of mechanism compromise occurs from certificate reissuance. Because a recipient can check expiration without referring to a CRL, and because termination of the mechanism compromise may occur without actual discovery of the compromise, the lag between discovery and termination is always zero.

Let

$CRL_{ca}$  denote the average time required for discovery to propagate from the discovering merchant to the CA CRL, and

$CRL_{mrch}$  denote the average time required for discovery to propagate from the CA CRL to the merchant communities' CRLs.

In the case that both parties detect compromise before year end, there are two possibilities:

- 1) The CA detects and propagates first, or a tie occurs, and
- 2) the merchant detects and propagates first.

$$P\left(\begin{array}{l} \text{time}(D_{ca}) + CRL_{mrch} \leq \\ \text{time}(D_{mrch}) + CRL_{ca} + CRL_{mrch} \end{array}\right) + \\ P\left(\begin{array}{l} \text{time}(D_{ca}) + CRL_{mrch} > \\ \text{time}(D_{mrch}) + CRL_{ca} + CRL_{mrch} \end{array}\right) = 1$$

### Types of Compromise Discovery

Information validation consists of one or more discrete events, in which the information underlying the subscription is checked for validity. The information validation event conducted as part of the subscription process is termed the *initial* information check. The periodic information check events that may be conducted after the subscription process are termed *subsequent* information checks. All subsequent information checks are assumed to have the same probability of detecting information compromise, and are further assumed to be evenly spread throughout the year being examined. Each information check subsequent to the compromise has ■ chance of detecting it; checks can only detect compromise after it has occurred.

### Information Compromise

Initial information compromise occurs at subscription time (time=0). An initial information check, at time 0, has a chance of detecting the compromise before certificate issuance occurs.

Let  $d_{init}$  be the chance of detecting initial information compromise at subscription time.

$$P_{det}(time = 0) = d_{init}$$

Detection may also occur subsequent to time 0, either because the CA failed to detect it initially, or because the compromise occurred after subscription time.

Let  $d$  be the chance of detecting compromise per check, and

$c$  be the number of information checks that occur before compromise.

Then the probability that detection occurs on the  $n$ th information check is:

$$P_{det}\left(check = n \mid \overline{Det(time = 0)}\right) = (1 - d_{subs})^{(n-c-1)} d_{subs}$$

The probability that detection occurs before the end of the year is

$$P_{det}\left(check \leq n \mid \overline{Det(time = 0)}\right) = 1 - (1 - d_{subs})^{(n-c)}$$

Let

$D$  be a discrete random variable denoting probability of detection,

$C$  be a discrete random variable denoting time at which compromise occurs

$\delta$  be ■ discrete random variable denoting whether detection took place before year end

Then we can determine the expected time until detection occurs.

$$E(D|\delta|C) = \sum_{i=1}^{365} \sum_{j=0}^1 \sum_{k=1}^{365} f(D|\delta|C)f(\delta|C)f(C)$$

$$f(C) = 1/365$$

$$f(\delta|C) = \begin{cases} (1-d_{subs})^{\text{int}(i/\text{period})-\text{int}(c/\text{period})} & j=0 \\ 1-(1-d_{subs})^{\text{int}(i/\text{period})-\text{int}(c/\text{period})} & j=1 \end{cases}$$

$$f(D|\delta|C) = \begin{cases} (1-d_{subs})^{\text{int}(i/\text{period})-\text{int}(c/\text{period})} d_{subs} & i > c, i \% \text{period} = 0 \\ 0 & \text{otherwise} \end{cases}$$

$E(D|\delta|C)$  is the expected day of the year at which discovery occurs. In the case that both the CA and the merchant discover compromise before year-end, the methods detailed earlier are used

$$\begin{aligned} E(\text{duration}) = & P(D_{ca})(1 - P(D_{mrch}))E(\text{dur}_{ca}) + \\ & (1 - P(D_{ca}))P(D_{mrch})E(\text{dur}_{mrch}) + \\ & P(D_{ca})P(D_{mrch})\min(E(\text{dur}_{ca}), E(\text{dur}_{mrch})) + \\ & (1 - P(D_{ca}))(1 - P(D_{mrch}))(365 - c) \end{aligned}$$

where

$D_{ca}$  CA detects compromise

$D_{mrch}$  merchant detects compromise

$\text{dur}_{ca}$  duration before CA detects compromise

$\text{dur}_{mrch}$  duration before merchant detects compromise

$c$  day on which compromise occurs

Whether discovery was accomplished by

Similarly for mechanism compromise, except that the period is the period of certificate reissuance, and we are determining time before termination of compromise rather than time before detection. Certificate expiration does not require consulting the CRL, so the time required for propagation of discovery is not needed.

### Detection by Merchants

A merchant may learn of a compromise from the CA via a CRL, or independently. Some examples include:

- The real subscriber complaining about fraudulent charges
- Failure to pay on the part of the malefactor
- Suspicious spending patterns (e.g. higher than normal)

and others. These are represented by two factors: a "base rate" chance to detect per event, and a chance based on the spending rate relative to "normal" users.

We can calculate merchant detection as with information compromise, with the following changes:

$$\text{period}=1$$

$$d_{mr} = d_{base} + (1 - 1/s^{sv})(1 - d_{base})$$

where

$s$  is the spending rate of the malefactor relative to the normal spending rate of this type of customer.

$v$  is a factor representing the effectiveness of the merchant's ability to detect compromise based on accelerated spending rates.

This formula represents the merchants' improved ability to detect compromise by examining out-of-pattern spending.

### Total Losses

Total costs are the sum of all direct costs incurred by the CA and merchants

$$\text{Issue} = \text{cost}_{issue} * \text{int}(365 / \text{period}) - 1$$

$$\text{Info} = \text{cost}_{init} + \text{cost}_{subs} * (\text{int}(365 / \text{period}) - 1)$$

$$\text{CRL}_{ca} = \text{cost}_{trans} * \text{rate}_{trans}$$

if the CRL is checked for every transaction

If batched CRL downloads are used, costs are negligible.

Total costs are

$$\text{Costs} = \text{Issuance} + \text{Information} + \text{CRL}_{ca} + \text{CRL}_{mrch}$$

Total losses are the sum of losses due to all types of compromise over the year, modified by detection times.

$$Losses = Rate_{spend} * \left( P_{Cinit} E(Dur_{Cinit}) + P_{Csubs} E(Dur_{Csubs}) + P_{Cmech} E(Dur_{Cmech}) \right)$$

Now we have both overall costs and overall expected merchant losses, and can compare the two.

### Parameter Assumptions

Table 1 provides the baseline parameter assumptions of the model for each of the three types of commerce.

The cost of operating the certificate infrastructure depends heavily on assumptions about how Certificate Revocation Lists are handled. Our model supports two modes of operation:

- 1) the merchant contacts the CA on every transaction to verify that a certificate has not been revoked
- 2) the CA sends a complete CRL to merchants once per day.

Assumption 1) results in much higher infrastructure costs for the CA but has only a modest effect on fraud losses. Thus, in the following we have assumed that CRLs are batched to merchants on a daily basis.

### Model Output

The model produces two primary outputs:

- 1) the cost per consumer of operating the certificate infrastructure for one year
- 2) the annual losses due to fraud normalized to a per consumer basis

From a design perspective, the goal is to choose design options which minimize the sum of these two costs.

### Model Analysis

Figure 2 shows how the cost of operating the CA infrastructure varies with certificate lifetime. Much of the cost of certificate issuance is the cost of validating the information in the certificate. For example, to validate a class 2 identity certificate, Verisign contacts Equifax to compare the information submitted by the applicant with data on file at Equifax[1]. We assume that the cost of this and other similar checks is on the

order of \$.75 per certificate for initial information validation. For very short certificate lifetimes, this information checking cost sharply increases the annual cost of the certificate infrastructure.

Figure 3 shows merchant losses with respect to certificate lifetime for various rates of consumer spending. Clearly the higher the average value of consumer spending, the greater the value of fraud losses on a per consumer basis. Merchant losses also vary strongly with the effectiveness of the merchant's ability to detect out-of-pattern spending through velocity checking. Figure 4 shows how the average time until a merchant discovers a fraudulent certificate varies with the effectiveness of his velocity checking schemes. If the merchant's scheme is effective, most malefactors will be caught in only a few days. Thus certificate lifetimes longer than this time have little effect on merchant losses.

Figure 5 shows the total of certificate issuance costs and fraud costs as a function of certificate lifetime and velocity checking effectiveness for high value purchases. If velocity checking is ineffective, certificate lifetimes should be kept as short as two weeks to limit fraud losses. If velocity checking is effective, certificate lifetimes of a year or more will minimize total costs.

In the case of microtransactions, because consumer spending is assumed smaller, the costs of fraud are less. Consequently, the issuance cost of shorter certificates dominates the fraud costs, and certificate lifetimes should be kept long no matter what the effectiveness of velocity checking.

The figures above have all assumed that a malefactor spends at a rate five times that of a normal consumer. However, a malefactor may adjust his spending rate after a certificate compromise to achieve the maximum possible return. In Figure 6 we show how total merchant losses vary with the malefactor's spending rate and certificate lifetime. If certificate lifetimes are long, a malefactor should spend at a rate similar to normal consumer's and avoid detection for the life of the certificate. If certificate lifetimes are short, the malefactor is better off spending as quickly as possible, despite velocity checking, before certificate reissuance terminates the compromise.

### Discussion and Conclusions

In this paper we have analyzed the costs and risks of a certificate infrastructure to understand the key variables which dominate total costs. Our results suggest that where certificates are used for consumer electronic commerce, there are two key design considerations which affect total costs: certificate lifetime, and the effectiveness of a merchant or payment system operator's velocity checking mechanisms for

detecting fraud. The analysis shows that optimal certificate lifetimes may be as short as a few weeks rather than a year or more as is the case in some current commercial CA infrastructure offerings.

While the results presented above focused on consumer use of certificates, the model can also be used for analyzing a much wider range of parameter assumptions than we have considered here. For example, if certificates are used in conjunction with bank transfers valued at hundreds of millions of dollars, one would assume very different values for the cost of certificate validation (more stringent methods would be employed) and the probabilities of information or mechanism compromise (private keys may be stored in secure hardware), but the model structure would remain largely the same.

We believe this effort has resulted in a clearer understanding of the relationships between the operational parameters of a commercial certificate system, and the risks and costs incurred by the participants. This research represents only a first step. There are many avenues for further work as a progressive development of this problem, as well as related issues. The existing model abstracts cost and risk shifting between the participants. Finding an optimal balance of cost and risk requires cooperation between the CA, merchants, and customers. Further, it may be possible to obtain empirical data with which to validate at least some of the issues under investigation, perhaps from a service provider or by analogy from the financial services industry.

## Acknowledgments

The author wishes to gratefully acknowledge the contributions of Marvin Sirbu, Doug Tygar, and Hadi Dowlatabadi.

Sponsored by the Air Force Materiel Command, under Advanced Research Projects Agency

Contract No. F19628-95-C-0018, "Electronic Commerce: The NetBill Project." Additional support was received from the National Science Foundation under Cooperative Agreement No. IRI-9411299. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency, the national Science Foundation, or the U.S. Government.

## References

[1] Verisign, Inc., "Answers to Frequently Asked Questions about Digital IDs", version 1.1 July 1995

## Bibliography

D.W. Davies and W.L Price. Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer, 2nd ed. John Wiley and Sons, 1989

Warrick Ford, Computer Communications Security: Principles, Standard Protocols, and Techniques. Prentice Hall 1994.

Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd ed. John Wiley and Sons, 1995.

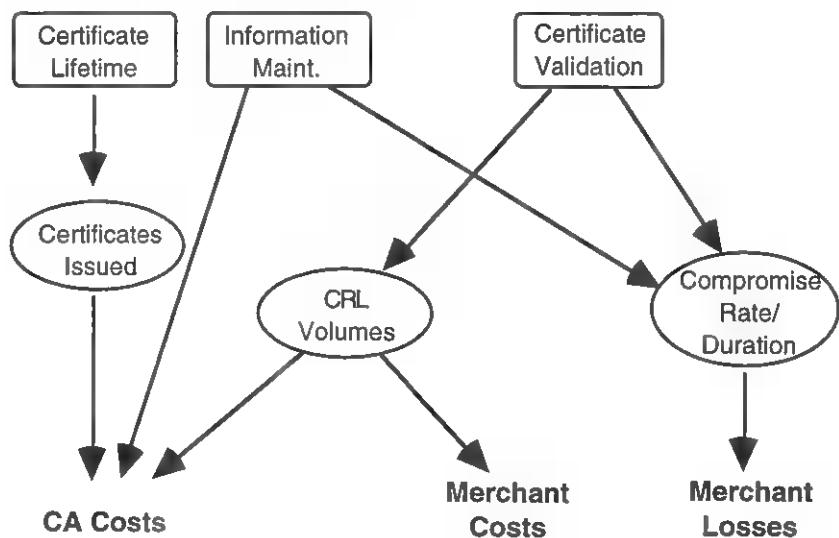
Marvin Sirbu J.D.Tygar, J.D., "NetBill™: An Internet Commerce System Optimized for Network Delivered Services". IEEE CompCon Conference, June 1995

American National Standards Committee X.9.55-1995: Public Key Cryptography for the Financial Services Industry. American National Standards Institute, 1995

IETF PKIX Working Group, Internet Public Key Infrastructure, draft. November 1995

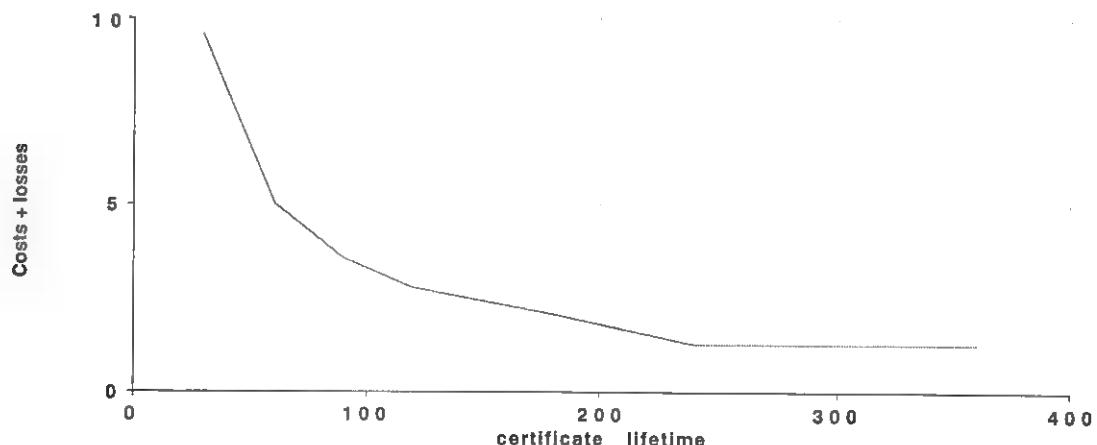
**Figure 1.**

**Diagram of Certificate Model**



**Figure 2**

**Cost varies with certificate lifetime**

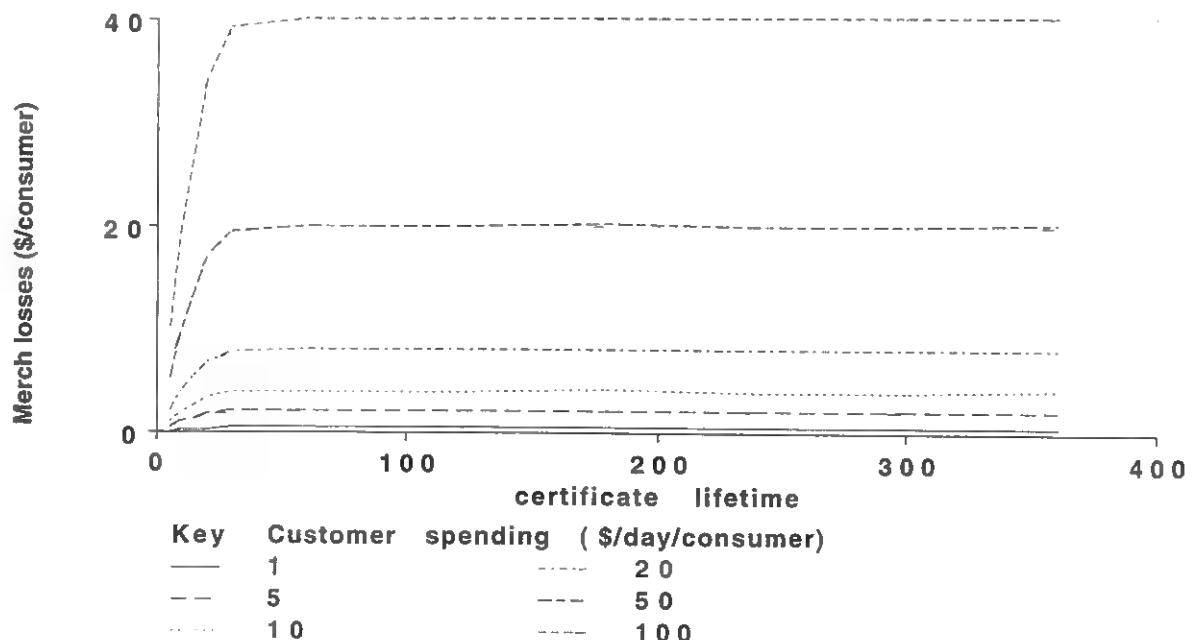


**Table 1.**

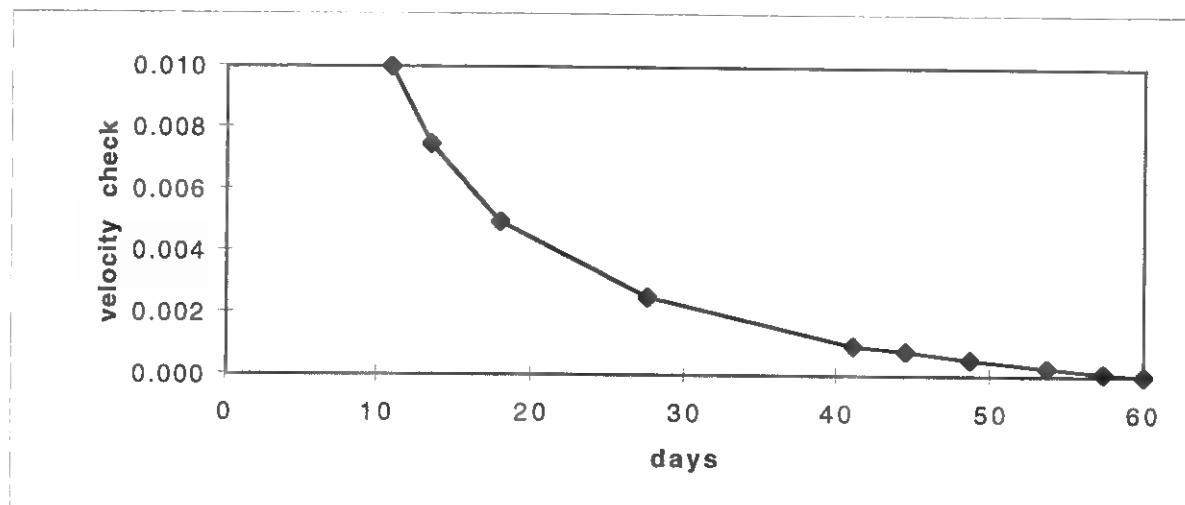
Variable	Value	Comment
Certificate lifetime	0.5 year	Critical design parameter
Information validation cost	\$0.75/cust	cost for validating info in certificate
Customer spending	\$1,5 or 100/day	Average value for each of the three electronic commerce scenarios
Customer transaction rate	0.25 trans/day	(\$/day) * (1/avg transaction size)
Mechanism compromise incidence	.001	Likelihood a consumer cert will experience mechanism compromise
Mean days until mechanism compromise	60 days	mean time until compromise given mechanism compromise occurs
Initial info compromise incidence	0.0075	Likelihood that submitted information for cert is false
Detect initial compromise	0.5	Likelihood that false information submitted for cert will be detected
Subsequent information compromise incidence	0.00025	Likelihood that subscriber information becomes false
Detect subsequent compromise	0.25	Likelihood that info compromise which occurs after cert issuance will be detected upon subsequent information check
Merchant base prob of fraud detection	0.0164	Prob merchant will detect compromised certificate on a given day given malefactor spending at average consumer rate. Set so that average time until detection is 60 days absent velocity checking.
Fraud spending multiple	5	malefactor spends at this multiple of average consumer spending rate
velocity checking effectiveness	.001	effect of malefactor's higher spending rate on likelihood of merchant detection
Terminate mechanism compromise	0.5	Likelihood that reissuance of a certificate will terminate mechanism compromise. <sup>1</sup>

<sup>1</sup> If a malefactor has planted a Trojan Horse on a consumer's machine, any newly issued certificate, even with a new key pair, may be immediately compromised.

**Figure 3**  
**Merchant losses per customer vs certificate lifetime**

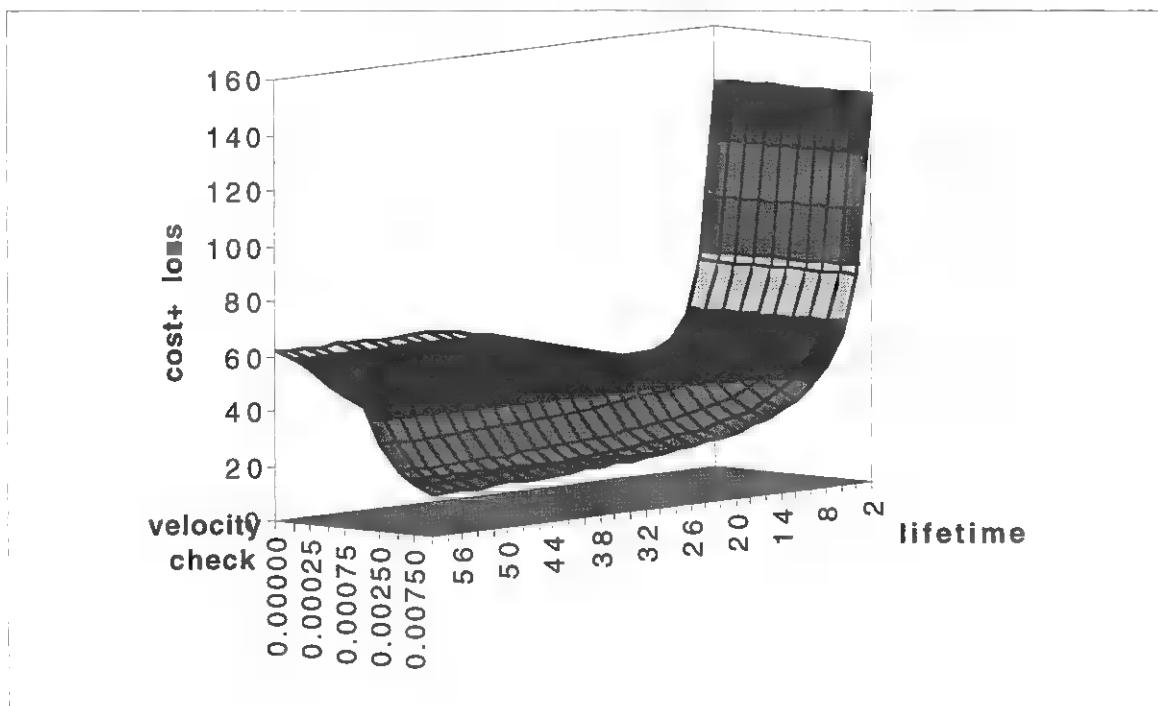


**Figure 4**  
**Expected Days Until Compromise Discovery versus Velocity Checking Effectiveness**



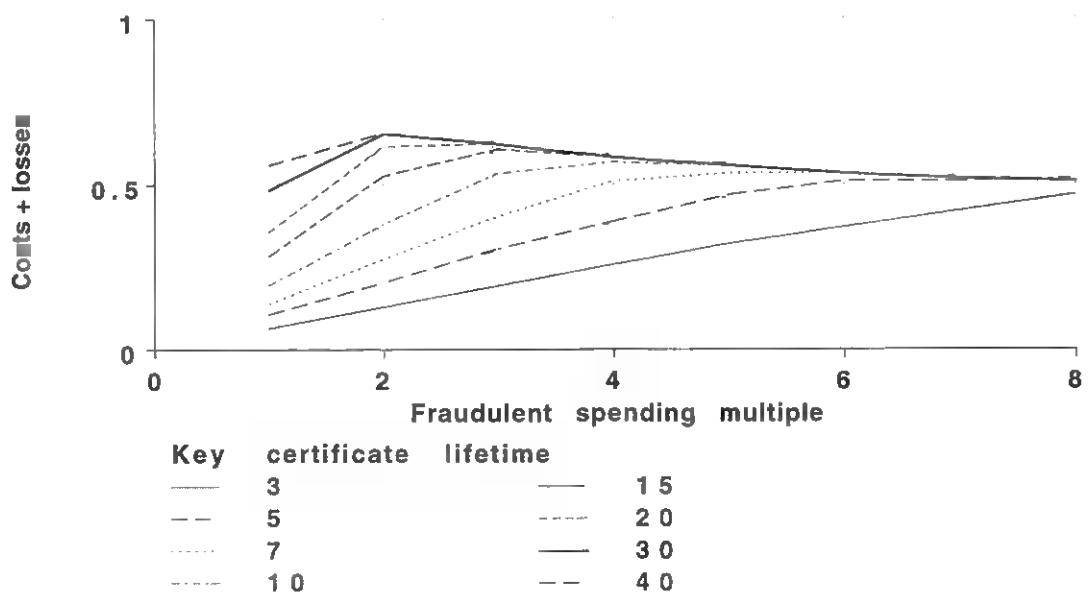
**Figure 5**

**Costs + Losses versus Certificate Lifetime and Velocity Checking Effectiveness**



**Figure 6**

**Costs + Losses versus Fraudulent Spending Rate and Certificate Lifetime**





# PAYMENT METHOD NEGOTIATION SERVICE

## *Framework and Programming Specification*

Alireza Bahreman  
*bahreman@eit.com*

Rajkumar Narayanaswamy  
*rajkumar@eit.com*

E-CO System Project<sup>w</sup>  
EIT/VeriFone

### ***ABSTRACT***

We propose a Payment Method Negotiation Service in this paper. This service enables transacting peers to determine and negotiate their common set of payment methods, protocols, systems or mechanisms, system providers, capabilities, accounts, and instruments. The service also enables the peers to select, among the negotiated choices, a particular payment system or mechanism to use to conduct commerce. We use a modular framework and define an object-based solution. The programming interface is modular and extensible and can interoperate with a wide range of payment systems. The service is transport independent and can support a variety of communication models. Therefore, all electronic commerce application developers would be able to accommodate this service in their applications. The Payment Method Negotiation service introduced here is part of the Generic Electronic Payment Services (GEPS) framework being designed and implemented in the E-CO System Project<sup>l</sup>.

## **1 Introduction**

The excitement surrounding electronic commerce has resulted in the introduction of numerous payment systems and mechanisms. Transacting peers must determine which of the many payment choices they will use in a commerce transaction. Due to this abundance of choice, end-users and applications alike face the challenges of negotiating and selecting a particular payment system or mechanism to use. Application developers face an increasing challenge of enabling such negotiation and selection with the introduction of a variety of new payment systems and mechanisms.

To address this problem, we propose a service called Payment Method Negotiation (MN). This service enables transacting peers to mutually agree on a common set of payment methods, protocols, systems or mechanisms, system providers, capabilities, accounts,

and instruments. The service also enables the peers to select, among the negotiated choices, a particular payment system or mechanism to use to conduct commerce. The service is generic enough to allow integration into a variety of applications and is extensible enough as to interoperate with a variety of payment systems. The MN service introduced here is part of the Generic Electronic Payment Services framework defined in a companion paper [GEPS].

To better understand what the *negotiation* and *selection* processes entail, consider the following examples. Consider the scenario in which Alice walks into a store. At the entrance, the merchant might have displayed the Visa®, MasterCard®, and Discover® logos. This is an indication that the merchant accepts those payment instruments. In a sense, the merchant is *offering* its payment capabilities to the customer upon entering the store. Alternatively, Alice could *query* the merchant

<sup>w</sup> This work has been funded by DARPA contract DABT63-95-C-0133.

<sup>l</sup> For more information, please visit our [WebSite] at <http://eco.eit.com>.

and inquire about acceptable payment mechanisms or methods. For example, Alice could ask the merchant if it accepts Cash or Debit as a method of payment. Note that neither the offer nor the inquiry necessarily presents a complete set of capabilities. In other words, neither party is obliged to reveal all of its capabilities. As a result of these and other related exchanges, Alice and the merchant will be able to determine if there are any acceptable means of payment which they both share in common. The above interaction describes the negotiation process. Only information specific to the payment choices are exchanged during negotiation. For example, the process does not include discount negotiation for using Cash versus Credit. That is outside the scope of this paper. Finally, the selection process is the determination of a specific choice of payment from the negotiated list of choices. For example, Alice might select one of her Visa cards.

The negotiation process does not require human interaction when implemented in the online world. This is because a list of payment capabilities can be stored online and retrieved automatically when needed. The Capability Management service of the GEPS provides this functionality. The only intelligence involved is in determining the amount of information to reveal at any stage of the negotiation. The selection process, on the other hand, will likely require input from the human end-user. The maintenance, and usage of, end-user's preferences could replace some of this interaction. (The Preference Management provides this service in GEPS.) However, the involvement of the end-user in the final selection process is very important. To not allow the end-user to select the final payment choice is like forcing Alice to give her purse to the merchant and letting the merchant pick the payment method of choice. If that happens, it is likely that Alice would not visit that store again, anytime soon!

The above example of negotiation and selection is possible using the Payment Method Negotiation framework suggested in this paper. In order to explain how we accomplish this, we organize this paper using the following structure. We start by presenting the terminology which we use throughout this paper in Section 2. We present the payment negotiation requirements in Section 3. Section 4 covers the Payment Method Negotiation framework while in Section 5 we introduce the programming interface and implementation details. We discuss the security considerations in Section 6. In Section 7, we compare our work to, and report on, other related work in this area. We wrap up with concluding remarks in Section 8.

## 2 Terminology

In this section, we present definitions for some of the terminology we use consistently throughout this paper. We provide this as an aid to understanding the discussions in this paper.

### 2.1 Payment Method

Payment method is a high-level categorization of a payment transaction. A way of making a payment, if you will. Some known payment methods include: Cash, Credit, Debit, Electronic Check, and Electronic Cash. For example, Alice might pay Bob ten US dollars in paper currency; she is therefore using the Cash payment method.

### 2.2 Payment Protocol

Payment protocol refers to a collection of messages, used to carry electronic payment related information and instructions among the parties involved, and the flow or sequence of those messages. As an example of an electronic payment protocol, consider the Secure Electronic Transaction (SET) specification.

### 2.3 Payment Mechanism or System

Payment mechanism or system refers to a complete system designed to enable and execute payment transactions among parties. Some payment mechanisms require a third party intermediary such as NetBill, from Carnegie Mellon University.

### 2.4 Payment System Provider

Payment system provider refers to the party operating a payment system or mechanism. For example, CyberCash, Inc., is a payment system provider, for their credit card payment mechanism.

### 2.5 Payment Capability

Payment system providers often implement the software needed to use their payment mechanism. Other developers then use this software in their application to conduct electronic payments. We refer to these implementations as payment capabilities. The nature of their implementation can vary (for example, an applet, a shared library, or even a plug-in). The end result, however, remains the same (i.e., it enables an application to use the services provided by a payment system).

## **2.6 Payment Service**

Payment service refers to a collection of payment functionality implemented as a module and used during a payment transaction by applications. There are five major payment services identified in [GEPS]. They are Payment Interface Management, Payment Method Negotiation, Preference Management, Capability Management, and Transaction Management. We dedicate this paper to covering the Payment Method Negotiation service.

## **2.7 Account and Account Proxy**

An account is where one gathers value. Alternatively, an account may provide its owner with the privilege to use a payment system. In the physical world, we are very familiar to the concept of an account (for example, a Bank account). In the online world, there may be new types of accounts, all with similar purposes as in the physical world. An account proxy, however, is the online representation of an account in the physical world. The account proxy can cache the information in the account (for example, the running balance). However, the account proxy's data is not necessarily up to date. One can close or destroy an online account, and an account proxy. However, one cannot destroy a physical account in a pure online fashion; in order to permanently close a physical account, one would require out-of-band mechanisms.

## **2.8 Payment Instrument**

A device or token used during the payment transaction. A particular Visa card with an account number and an expiration date is an example of a payment instrument to use in transactions of type Credit payment method.

# **3 Requirements**

The payment negotiation process must meet the following requirements. This set of requirements is a subset of those covered in a related document [NegReq]. For a comprehensive set of requirements related to Payment Negotiation, please review that document. Here, we will express a few of those requirements that we feel are important and which we have addressed by our framework. We group these requirements under business, functional, and technical categories.

## **3.1 Business Requirements**

The business requirements ensure the successful adoption of the payment negotiation mechanism by

application developers and end-users. These dictate that, the negotiation mechanism must require the least amount of change from the existing systems; it must be easy and cheap to use and integrate with. In addition, the negotiation mechanism must facilitate payment system competition without stifling innovation. It should be convenient for customers and merchants to use (must particularly be easy to use for the customers). Finally, the use of the negotiation mechanism must ultimately drive down the cost of doing business

## **3.2 Functional Requirements**

The functional requirements dictate that the negotiation mechanism must support all payment systems and mechanisms; provide for a smooth transition from pre-to post-negotiation (i.e., be well integrated); and be easy to use for both purchasers and merchants. In addition, the negotiation model must support both automated and manual selection.

## **3.3 Technical Requirements**

We group the technical requirements into three sets: message format, protocol, and infrastructure requirements. The message format requirements dictate that the message format be extensible in order to support all payment systems. It must also allow for the conveyance of payment related auxiliary information. The messages must support mechanism to specify preferences and their strength (for example, optional, required, or refused). The message format must be transport independent (i.e., must be possible to carry it over as binary data as well as ASCII). As for the protocol requirements, the design of the payment negotiation must be symmetric, meaning that both the customer and the merchant can initiate the negotiation. It must also be transport mechanism independent, which in the case of a protocol means that it can be accomplished in both online and store-and-forward fashions. The protocol must also provide support for instances in which there may be proxies or intermediaries between the negotiating peers. The infrastructure requirements dictate the use of a framework such as GEPS in conjunction with the payment negotiation. This way, the negotiation module can take advantage of the integrated and extensible capability list management, and flexible preference management to accomplish its tasks.

## 4 Framework

### 4.1 Functionality

The Payment Method Negotiation (MN) service enables transacting peers to mutually agree on a common set of payment methods, protocols, systems or mechanisms, system providers, capabilities, accounts, and instruments. The service also enables the peers to select, among the negotiated choices, a particular payment system or mechanism to use to conduct commerce.

The MN service is generic enough to allow integration into a variety of applications and is extensible enough as to interoperate with a variety of payment systems. There are three reasons for this. First, the negotiation logic may occur at any time during an application process. For example, parties can negotiate just prior to a payment. Alternatively, parties can inquire into the other peer's capabilities at any time. For example, the customer and merchant may wish to determine if they share any payment capability before beginning the shopping process. This is to ensure that they will not fail to complete the payment transaction later and waste all the effort made during the shopping. Second, either party can initiate the negotiation resulting in a symmetric process. Either the customer or the merchant (client or server in a distributed application) could initiate the negotiation process. Third, the MN service is extensible and allows for the integration of all payment systems. The MN protocol uses extensible message syntax and can carry relevant information on all payment systems between peers.

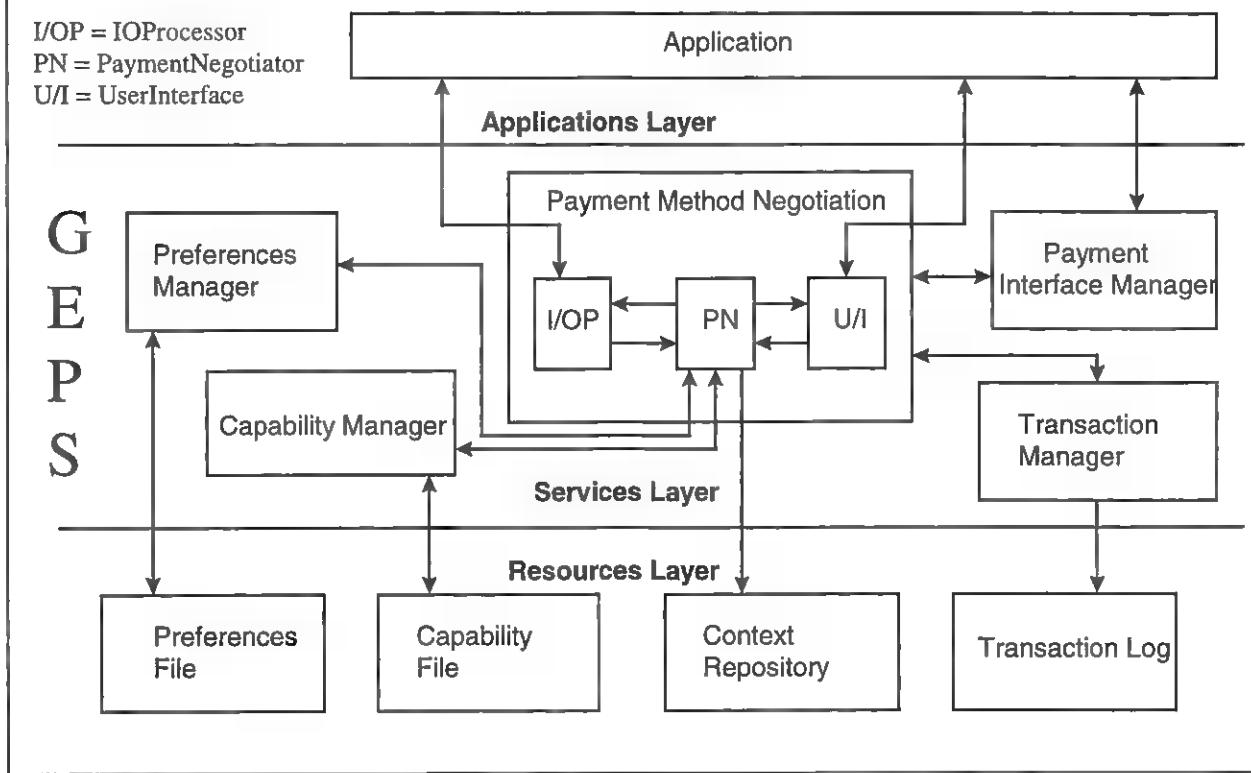
The MN has three components—IOProcessor (I/O/P), PaymentNegotiator (PN), and UserInterface (U/I). The next section explains, in greater detail, the implementation of each component. Figure 1 depicts the MN framework and components. The framework calls for each of the transacting peers to have their individual installation of the MN service. In order to conduct the negotiation, the MN uses peer-to-peer communication between these two installations through exchanging tokens. The applications that use the MN service are responsible for the transport of these tokens. The content of the tokens is opaque to the applications.

The I/O/P translates MN tokens into a representation that the application can transport and vice versa. There can be several implementations of this object leading to support for different transport mechanisms. Using this extensible mechanism, we have made the MN service transport independent. Section 5.2.1 provides an example of the use of this mechanism.

The PN maintains the state of the negotiation and implements the negotiation logic and policy. It is the primary object responsible for making the negotiation possible. The PN maintains and enforces the negotiation policy. Different implementations of PN are necessary to implement varying policies. There are four basic phases during the negotiation process. They are Query, Offer, Process, and Selection. In the Query phase, a party is asking the other for a listing of its preferred payment capabilities. In other words, it is asking the other party to list its acceptable payment mechanisms. Either party can initiate the negotiation by entering the Query phase. During the Offer phase, one party is listing its capabilities to the other party. A party can begin the Offer phase either as a result of a query, or voluntarily. Either party can initiate the negotiation by entering the Offer phase voluntarily. Note that in all cases, neither party is obliged to reveal all of their capabilities to the other. The policy implemented by the PN will dictate the level of openness in the exchange. During the Process phase, the parties will exchange messages back and forth in order to determine a set of common and acceptable payment capabilities. At the end, there may be one or more such capabilities identified. For each capability, the parties can negotiate acceptable usage criteria or hash specific details of that capability. For example, the parties might start negotiating on what capabilities they share that support the Credit payment method. Once both parties determine that they have such a capability, the parties might narrow the choices by listing the payment instruments used in a Credit transaction. This might further reduce the number of choices. Narrowing the choices in the negotiation might require some information other than those directly specific to payments (for example, profile information). For instance, one of the parties involved in the negotiation might need the residence country of the other to choose a payment mechanism. We achieve this by passing such information as additional details for the use of a payment capability. If there is no common payment mechanism or capability, the negotiation is over and the parties cannot complete the payment transaction. If there are several common capabilities, then the parties enter the Selection phase to select one of those choices as the means for payment. If only a single common payment capability is negotiated, the selection is automatic, but may still require end-user confirmation.

The U/I provides a consistent way to obtain end-user input when needed. The goal of the MN service is to make the negotiation process as automatic as possible (i.e., with minimum interactions required from the end-user). This is partly to avoid end-user confusion and

**Figure 1- Payment Method Negotiation service in relationship to GEPS framework**



partly for ease of use. The end-user does not always know all of the intricate details regarding payment capability choices. For example, while the end-user knows the difference between their Visa card and their Discover card, the end-user might not be aware of the differences between SET and CyberCash (two different payment protocols). It is easier for the end-user to make one choice at the Selection phase instead of being called upon several times during the Process phase. The U/I provides the end-user input, when required. In most cases, the Selection phase will require the final end-user input to complete the negotiation with the selection of a payment capability.

#### 4.2 Relationship to GEPS

Figure 1 illustrates the schematic of the MN service and how it relates to the other modules in the Generic Electronic Payment Services [GEPS] framework. The MN service introduced here is an integral part of the GEPS framework. Both negotiating peers (the server and the client) will have a GEPS environment implemented. The input message to the PN is a negotiation token wrapped in a transport specific format. The I/OP converts the input into a negotiation token. The PN object processes the token using the preferences given by the Preference Manager over the

list of payment capabilities generated by the Capability Manager. The PN may also need input from the end-user. In which case, the U/I opens an application level window (part of the Payment Interface Management's Wallet) to indicate the status to the end-user and/or receive input from him/her. The I/OP receives the output from the PN (a negotiation token), bundles it in a transport dependent form, and delivers it to the application for transmission. The Transaction Manager might log individual transaction messages and operations for later use in trace and audit trail operations.

## 5 Implementation

In this section, we present a more detailed look at the Payment Method Negotiation framework by introducing its programming building blocks. To present the Application Programming Interface (API) of the MN service, we describe the main classes and their methods.

### 5.1 Resources

The MN uses a set of data-types (NegotiationContext and NegotiationToken) in order to implement its services. These resources help in maintaining negotiation state and conveying negotiation messages.

### 5.1.1 NegotiationContext

The context repository stores an object of this type for every session. It contains information about the buyer and seller profiles, details of the transaction, list of payment capabilities (both rejected and accepted), and any additional payment related information. A session may include several transactions. Once the payment capability of choice has been negotiated, all transactions in that session can use the results. Alternatively, each transaction could initiate a negotiation, possibly building upon the results of any previous negotiation made in that session.

### 5.1.2 NegotiationToken

This object contains the negotiation information and information about each stage in the payment negotiation process. A representation of this object travels on the wire between the negotiating parties. Applications do not know anything about its content. Only the peers at the MN service level know how to process these tokens. Specifically, only the PN module uses these tokens. Each NegotiationToken has a *type* field that indicates what processing function can the recipient perform on that token. The valid types already defined for NegotiationTokens include: Inquiry, Request, Intermediate, Final, Selected, Failed and Invalid. The use of these types will become clear in Section 5.2 when we explain the processing functions. The I/O module also is able to parse these tokens as it has to wrap (unwrap) them in (from) application specific format.

We define additional data types to hold transaction related details, such as payment amount information and party profiles. There are also utility functions that provide information about the NegotiationToken and NegotiationContext. For example, there will be a function to extract the final selected capability or the list of negotiated payment capabilities from a NegotiationToken (i.e., `get_payment_capability` and `get_capability_list`). See Appendix A for more details.

## 5.2 Processing

The MN service is part of the GEPS framework. From a structural perspective, it consists of three components—I/O, PN, and U/I. Appendix B contains more details on each of the classes implementing these components.

### 5.2.1 IOProcessor (I/O)

This component takes care of the interface between the applications and the MN tokens. Recall that the MN uses peer-to-peer communication by passing tokens.

The application is responsible for transporting these tokens. The I/O module enables the wrapping and unwrapping of the NegotiationTokens generated by the MN into a format suitable for the applications to transport. The I/O functionality includes receiving input messages from the transacting peer, converting them into NegotiationTokens, and embedding negotiation intentions from NegotiationTokens into outgoing messages. This is an abstract class, which could be subclassed to work with different transport protocols. Functions needed by the I/O include:

**to\_external\_form**: converts the input token (NegotiationToken) into a message that is suitable for transport (TransportToken) to a remote peer by the calling application; and

**to\_internal\_form**: converts the input message (TransportToken) into a NegotiationToken that is suitable for internal processing.

The ability to subclass the I/O provides the means to achieve transport independence for the MN service. We illustrate this by an example using a subclass of the I/O intended for use within the HTTP transport and specifically using the Protocol Extension Protocol (PEP). We call this class HTTP-IOProcessor, to denote that it is a derived subclass of the I/O.

In order to understand the example, we must first give a brief summary of the message formats in the PEP protocol. For detailed information, please refer to the documentation [PEP]. Essentially, all PEP messages are HTTP headers to be exchanged between a client and a server. There are four header types: Protocol, Protocol-Request, Protocol-Query, and Protocol-Info. The Protocol header indicates the extension protocol used in that message, while the Protocol-Request expresses the interest in a protocol in the response message. The Protocol-Query header allows one party to ask the other if it supports a particular extension; the response comes back in a subsequent message using the Protocol-Info header. The value for each of these PEP headers is one or more bags. The bag syntax is a curly parentheses-delimited *bagname* and *bagitem* pair. The bagname is a name of a protocol used in creating the bag. The bagitem can be one or more bags leading to a recursive structure. The bagitem could also be null or a token.

Using the PEP syntax, it is easy to construct a message containing all the information needed in any protocol (for example, a payment negotiation protocol). Specifically, the Universal Payment Preamble [UPP] is the vocabulary used to encode payment related

information over PEP. This is indeed the approach taken by the Joint Electronic Payment Initiative (JEPI) project. In the E-CO System project, we have adopted that same syntax in encoding the necessary information for transport using HTTP-IOPProcessor.

The HTTP-IOPProcessor would have to parse HTTP headers using the PEP extension and translate the value of those headers into a NegotiationToken object when receiving data from the peer. When sending data to the peer, the reverse process is used (i.e., the NegotiationToken data are converted into bags suitable for transmission using the PEP headers).

For example, a sample input to the HTTP-IOPProcessor (subclassed from IOP) employed in an HTTP/PEP combined environment could be a header like:

Protocol-Request: {GEPS-MN Bag-A Bag-B Bag-C}

The output would be a NegotiationToken, representing the negotiation inputs {Bag-A} {Bag-B} {Bag-C}. The PN object processes this token. Similar data conversion happens when the input is a NegotiationToken; the token is converted into one or more PEP header(s) to be embedded in HTTP messages.

### 5.2.2 PaymentNegotiator (PN)

The PN is the primary component of the MN that is responsible for understanding the negotiation requests and generating appropriate responses. The PN creates a NegotiationContext and stores it with an appropriate handler in the context repository for every transaction session. Once assigned, the handler information travels in all future NegotiationTokens generated in that session. The PN receives a NegotiationToken as input, performs a series of steps, and outputs another NegotiationToken and a status value. As a side effect, the PN also updates the NegotiationContext. The status indicates if the negotiation is completed, failed, or needs to continue (i.e., COMPLETED, FAILED, and CONTINUE). Some of the functionality performed by the PN are enumerated below:

- Obtain the corresponding context from the database; if there is none, create a new context and assign it the handler from the NegotiationToken.
- Receive list of capabilities from the current context and NegotiationToken.
- Validate the capability list with the help of GEPS Capability Manager service.
- Process end-user's preferences with the help of GEPS Preference Manager service.

- Prompt the end-user, in case of difficulty processing negotiation decisions.
- Create an output NegotiationToken, representing negotiation intentions of the end-user.

There are five main methods of the PN which when called in different sequences allow applications to perform all possible payment negotiation scenarios. In Section 5.3, we explain these scenarios in greater detail. We explain these five methods below:

***begin\_negotiation\_context***: One party calls this function to create the initial NegotiationContext and NegotiationToken thereby beginning the negotiation dialog. There is no need for an input NegotiationToken to make this call. If succeeded, the function generates a NegotiationToken of type Inquiry, which does not contain any payment capability list. In the event of failure, the output token type is Failed. The only valid status codes generated from a call to this function are FAILED or CONTINUE. The former indicates that the function failed to process the call, while the latter indicates that the negotiation must continue.

***offer\_negotiation\_context***: Another way to begin the payment negotiation dialog is with a call to this function. There is no need for an input NegotiationToken to make this call. If successful, it generates a NegotiationToken, of type Request, that contains a list of all valid payment capabilities that exist in the caller's GEPS environment. These capabilities may or may not be fully detailed yet. If this function fails, it returns a token of type Failed. Valid status codes generated are FAILED and CONTINUE. The application policy might dictate a degree of exposure that is more restricting than that enforced by the current implementation. In that case, a different implementation of the PN must be used.

***process\_negotiation\_context***: This is one of the most commonly used functions during the negotiation dialog. This function requires a valid NegotiationToken as input. Only tokens of type Inquiry, Request, and Intermediate are valid as inputs to this function. The function either delivers a list of applicable payment capabilities, or returns an error condition (if the transaction is not possible due to a lack of common payment capabilities). Valid status code returns are FAILED or CONTINUE. The output token is either of type Intermediate (meaning that the token includes a list of payment capabilities), or of type Failed (meaning that the negotiation has failed). The capability list included in the output token includes all of the payment capabilities of the input token (marked

rejected or accepted possibly with additional detail). It also includes any additional payment capability the caller wishes to offer for negotiation. This function uses the NegotiationContext as well as the GEPS Capability Management (CM) service to process the list of capabilities. It uses the CM to reject or accept (possibly with additional detail) the payment choices. The NegotiationContext records the result of the negotiation; it is used to avoid duplicate processing by the CM (i.e., if a payment capability has already been rejected, it will not be further processed again). This function probes all local payment capabilities that have not been previously rejected to offer their capability for inclusion in the outgoing payment capability list of the output NegotiationToken. Since both parties maintain the negotiation context locally, there is no need to repeat any payment capability in the token's list (unless there are changes).

**finalize\_negotiation:** The use of this function indicates the caller's intention to end the negotiation while still offering the recipient the opportunity to make the final selection. The recipient must respond by selecting one of the payment capabilities listed in the output NegotiationToken generated by this function or end the negotiation with a token of type Failed if none are acceptable. Valid input tokens are of type Intermediate. The resulting token is of type Final or Failed. The GEPS Preference Management service and the user's preferences help sort the capability list in the output token; the list includes all payment capabilities negotiated so far. The valid status codes generated as a result of this call are FAILED or CONTINUE.

**accept\_negotiation:** Either peer uses this function to end the negotiation or respond to a finalize\_negotiation call by the other peer. The function selects a common payment capability that has been negotiated so far. Only tokens of type Final or Intermediate are valid inputs. If successful, the output token is of type Selected and includes the selected payment capability. Otherwise, the token type is Failed. Valid status codes generated are COMPLETED or FAILED.

When necessary, the functions above will use the U/I module to prompt the end-user for some input to help make negotiation decisions. All functions process the input tokens using the corresponding negotiation context from the context repository and update the context as well. All functions also generate error messages if their processing fails.

### 5.2.3 UserInterface (U/I)

The U/I component allows the PN to receive input from the end-user when needed during the negotiation phases. It also ensures that there is a consistent interface to the end-user which ultimately leads to ease of use and acceptability of the MN framework. Not all applications could use the same graphical user interface. Therefore, we have specified the U/I module as an Interface, so that the applications can independently implement it providing their own unique look and feel. (In the GEPS framework, the Wallet in the Payment Interface Management service implements the U/I.) At the very minimum, however, all implementations must support the following functionality.

**get\_choices:** Allows the end-user to identify a list of choices, possibly ordered according to their preference. For example, one can ask the end-user to place a check mark against all acceptable payment methods. The end-user might choose Cash and Credit. If ordering is allowed, the end-user might place the number one next to Cash and the number two next to Credit. This indicates that while the end-user would like to negotiate both Cash and Credit payments, the end-user prefers Cash. The Preference Management service of the GEPS framework can automate the ordering aspects.

**select:** This helps the end-user select a payment capability from a list of choices. This is used during the Selection phase. For the most part, this is not automated in order to empower the end-user. In some cases (as in small value transactions) it may be automated using the end-user's preferences and the Preference Management service of the GEPS framework.

**fill\_details:** This is a mechanism to ask the end-user to fill in the details for a payment capability. For example, to determine the taxes on a Credit transaction one might require the shipping address. The Wallet functionality of the GEPS Payment Interface Management service can automatically provide most of the required data.

**confirm:** This is used to ask the end-user to confirm a payment choice or to authorize the payment transaction.

**display\_message:** This is a generic functionality to display a message to the end-user.

**add\_system\_messages:** This function is used to display progress status to the end-user with visual effects. It could also be used for transaction logging.

Each of the methods presented above must allow the end-user to abort the transaction at that point. The application developers may provide additional methods to give the end-user more control over the negotiation process.

### 5.3 Message Flow

What we have described so far is the foundation and framework for the Payment Method Negotiation service. The resources (`NegotiationContext` and `NegotiationToken`) and the MN building components (the I/OP, PN, and U/I) presented here describe the implementation essence of MN. To help place things in perspective, it is instructional to see how two parties can negotiate and select a payment capability using these basic primitives. How these primitives support a variety of possible negotiation and selection scenarios? When can one use the U/I functionality? How do these primitives support the Offer, Query, Process, and Selection phases of negotiation? Finally, what is an example of the sequence of function calls in a typical negotiation and selection process?

We hope to be able to answer these and similar questions in this section by providing an example of a simple negotiation and selection between Alice and Bob. We discuss each stage of the process at length along the way. We strongly believe that specifying the message flow is essential to understanding the Payment Method Negotiation service (that is, we need more than just specifying the message syntax or the API).

There are numerous possible scenarios one could imagine between two parties, Alice and Bob. We describe one such message flow between Alice and Bob in stages (see Figure 2). Since the MN is symmetric, it is not important to distinguish between the party roles. Therefore, either Alice or Bob could be the customer (or the merchant). We assume they are both negotiating peers running the GEPS environment on their systems. They both have an implementation of the MN service as part of their GEPS implementation. They might each have different negotiation policies implemented. We also assume that each party has independently acquired and installed one or more payment capabilities on their system.

#### 5.3.1 Stage One: Initiate Negotiation

Let's begin the example with Alice initiating the payment negotiation process. There are still several possible scenarios caused by two variables. The first variable is with respect to the length of the negotiation

itself. Alice could just be curious and would like to know Bob's payment capabilities; that is, she is not at the point where she must pay Bob. She could wait until she reaches that point later, but she has decided to make sure early on that Bob is able to receive her payment. The difference is in the criticality of the time needed to complete the negotiation process. If one urgently needs the result, the number of messages they can afford to exchange is less than the case in which there is no urgency.

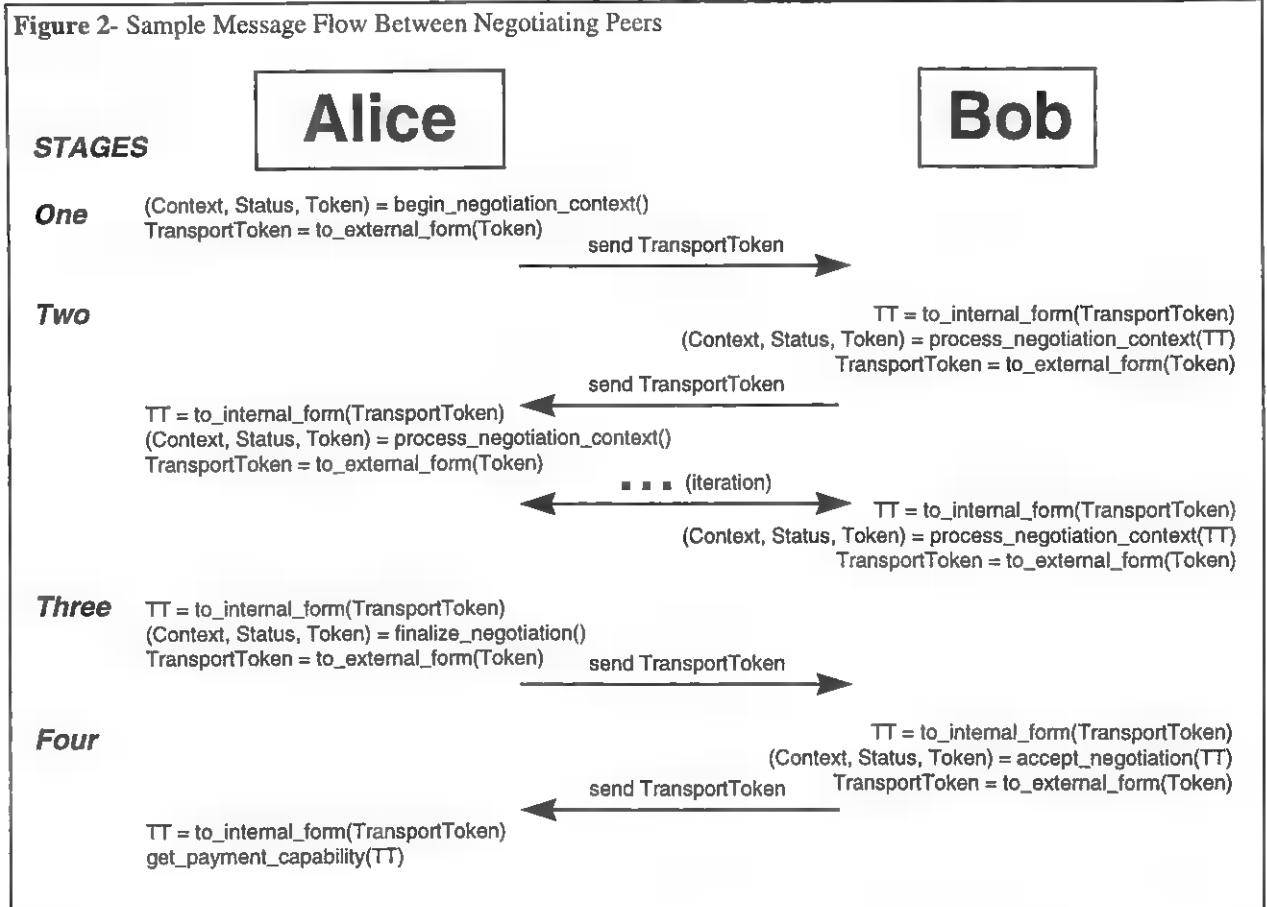
The second variable is with respect to the amount of information revealed by the party who starts the negotiation. Alice can initiate the process in one of several ways: she could offer all (or some of) her payment capabilities and ask Bob, which of those he supported; she could simply ask Bob what he supported without revealing any of her capabilities; and she could simply pick her favorite payment capability and offer it to Bob and wait and see what his response is. The function calls that can support these are `offer_negotiation_context`, `begin_negotiation_context`, and `offer_negotiation_context`, respectively. The difference is in the amount of information being revealed by the party who goes first.

The above scenarios are examples of the Offer and Query phases. In our example, we have chosen the scenario in which Alice asks Bob to reveal his payment capabilities without revealing any of her own. This is depicted as stage one in Figure 2. In our example, we assume that the negotiation is started well in advance of the actual payment. Therefore, there are no immediate limits on the number of the message exchanges that can be afforded.

#### 5.3.2 Stage Two: Processing Negotiation

This stage corresponds to the Process phase. It could involve as many message exchanges as the two parties are willing to engage in. Each time, one party offers some or all of his/her capabilities for the first time; rejects any of the capabilities of the other party (offered in a previous message); or enumerates additional detail on a previously offered payment capability. Since each party maintains the negotiation context, it is not necessary for the parties to repeat themselves by resending already negotiated payment capabilities unless they have been changed. The negotiation messages can be piggybacked on other application communications that the parties might be engaged in. For example, the messages can be exchanged while a customer is browsing a web-based online catalog.

**Figure 2- Sample Message Flow Between Negotiating Peers**



In our example, this is the first opportunity to use the U/I in order to make the negotiation process more efficient. For example, the PN can more efficiently process the negotiation if it first found Alice's payment method of choice. Assuming that her choice is to pay with a credit card, one only needs to negotiate Credit payment capabilities. This avoids the time spent negotiating over other payment methods.

### 5.3.3 Stage Three: Finalizing Negotiation

This stage represents an attempt by one party, Alice, to put an end to the Process phase. However, she is still allowing Bob to make the final selection. Alice uses the `finalize_negotiation` function to stop the negotiation loop started in the previous stage. This still corresponds to the Process phase as neither party is making a selection. The ability for either party to call the `finalize_negotiation` function supports numerous negotiation scenarios empowering both parties.

One can also use the U/I in this stage to allow the end-user to approve the use of payment capabilities negotiated so far or even manually sort them in their preferred order. Most of this, of course, could be done

automatically as well using the GEPS Preference Management service.

### 5.3.4 Stage Four: Selecting Payment

Another way to put an end to the negotiation process is to enter the Selection phase by calling `accept_negotiation`. This is either in response to a `finalize_negotiation` from the other party, or voluntarily. Either way, one party is taking the initiative to make a selection therefore ending the negotiation process. The ability for either party to call the `accept_negotiation` function supports numerous negotiation scenarios.

To allow the final selection, one would most likely use the U/I. Except for cases such as small value transactions, in which the preferences might dictate an automated selection. Also, of course, if the party making the call is a server and does not represent a human, the selection might take place without the help of the U/I.

The example above and the PN functions assume that there will be a number of messages exchanged between the parties during the Process phase and at least one for

completing the Selection phase. In fact, there is no predefined maximum number of exchanges. If the application requires a minimum number of exchanges to process negotiation and selection, it could call the following sequence of function calls instead. Alice would have to initiate with an `offer_negotiation_context` function that fully describes her desired payment capability or capabilities. Bob follows by calling the `process_negotiation_context` function returning only the selected payment capability. In this example, both parties must be aware that the intention is to minimize the number of messages exchanged. To programmatically enforce and convey this behavior, Alice must instead begin the negotiation with a call to the `finalize_negotiation`. We have decided not to allow the use of `finalize_negotiation` as the first function called in the negotiation process. This is in order to avoid overloading the function with the initialization logic performed in the `begin_negotiation_context` and `offer_negotiation_context` function calls.

#### 5.4 Policy Implementation

Another very important aspect of the payment negotiation to consider is the policy implementation. To fully describe the negotiation process, one must do more than just specify the message syntax, or the message flow. One must define the various policies that impact the message flow and to understand how to implement those policies. We devote this section to a discussion of the relevant policies and how and where in the system to implement them. There are a number of policy issues that we can identify. Table 1 summarizes the results. For each policy issue, it describes where to implement ■ solution—at the calling application level, at the PN level, or at the U/I level.

Overall, we believe in not following the complicated path of policy choices. That path will likely require end-user policy management as another GEPS service. We instead have decided to concentrate on ■ few limited policies enforced as different implementation of the PN and the U/I. This gives us a valuable experience on which to build future generic policy management services.

##### 5.4.1 Negotiation Sequence

This issue deals with the sequence of negotiation functions being called by the application. This affects the length of the negotiation process and the end-points. Using the tokens' type field, we can encode part of the intended negotiation logic. For example, the token generated as ■ result of the `finalize_negotiation` call is of type Final. This ensures that the receiving party

calls accept\_negotiation on that token. The type field effectively controls what function to call in order to process what type of tokens. In addition to the token type field, the calling application must be aware of the negotiation process. Through making the function calls, an application can initiate, process, and terminate payment negotiation. For example, unless the application is aware of the negotiation process, the two parties could loop forever each calling the function `process_negotiation_context`. The token type is not sufficient to break the loop; the application must call either the `accept_negotiation` or the `finalize_negotiation` function. `NegotiationContext` maintains a counter that helps applications understand the negotiation progress. During every session, applications can use the counter to determine the length of the negotiation process.

##### 5.4.2 Negotiation Efficiency

The goal of the negotiation process is to determine common payment methods, protocols, systems or mechanisms, system providers, capabilities, accounts, and instruments. To improve the efficiency of the negotiation, the peers can negotiate in that order instead of any random order. This is because the above order ensures that the parties will not waste time negotiating on payment capabilities that they do not have or wish to use. For example, assume that Alice wants to pay using her credit cards. Had the parties first agreed on the payment method, they would not have wasted time negotiating all common payment instruments and protocols of type Cash method. Even if they both agree on the payment method of Credit, they should not negotiate the payment instrument (i.e., what specific credit card) before they first negotiate a payment protocol to conduct the transaction in. If one party has the SET protocol implementation while the other has the CyberCash credit card payment, they will still not be able to perform the transaction. Both the GEPS Preference Manager and the MN's U/I are involved in allowing the end-user make his or her preferences known early. The application has the ultimate control in determining the order of the choices to negotiate.

##### 5.4.3 Exposure Control

By making an offer, each party is revealing their payment capabilities. It might be against one's policy to reveal all their payment capabilities at any time. Either party might want to reveal the minimum amount of required information. Also the party who starts first, will be the first to reveal their capabilities. There are two control mechanisms implemented with respect to this issue resulting in different policies. First, either party can decide to be the first to reveal by calling the `offer_negotiation_context` function. Second, using

**Table 1 - Policy considerations and where in the framework they are implemented and/or enforced**

Policy	Implementation	Implementation	Implementation
Negotiation Sequence	<ul style="list-style-type: none"> <li>■ Sequence and order of function calls</li> <li>■ Terminate negotiation loop</li> </ul>	<ul style="list-style-type: none"> <li>■ Token type field enforces use of functions</li> <li>■ Counter maintained in the NegotiationContext</li> </ul>	—
Negotiation Efficiency	<ul style="list-style-type: none"> <li>■ Specify the order of negotiating choices when making a function call</li> </ul>	<ul style="list-style-type: none"> <li>■ Follow orders provided by the calling application</li> </ul>	<ul style="list-style-type: none"> <li>■ Ask payment method first if not specified in the preferences</li> </ul>
Exposure Control	<ul style="list-style-type: none"> <li>■ Choice of revealing first by calling offer_neg... instead of begin_negotiation_context</li> </ul>	<ul style="list-style-type: none"> <li>■ Different implementations of PN provide various exposure policies: e.g., all at once or gradual</li> </ul>	—
Preference Ordering	<ul style="list-style-type: none"> <li>■ Specify preference criteria used ordering</li> </ul>	<ul style="list-style-type: none"> <li>■ Order by preference criteria</li> <li>■ Sort using GEPS Preference Management</li> </ul>	<ul style="list-style-type: none"> <li>■ When preference not specified, ask end-user</li> </ul>
Solicit Input	<ul style="list-style-type: none"> <li>■ Application implements the U/I and controls policy</li> </ul>	<ul style="list-style-type: none"> <li>■ Call U/I functions for all end-user input</li> </ul>	<ul style="list-style-type: none"> <li>■ U/I implemented by the application</li> </ul>

different implementations of the PN functions, one can control the amount of exposure. Some will reveal all choices when probed, while other implementations might hold back on some information and apply a gradual release of information.

We also considered the idea of maintaining *filters* by the PN. The applications implement these filters and use them to monitor and control the exposure level. For example, a filter might remove half of the items in a token's payment capability list before passing that token to the other peer. By implementing the filters, the applications could enforce all the policies they would like. We have decided that, at least initially, the use of different implementations of the PN would be sufficient to achieve the same objectives while being more realistic.

#### 5.4.4 Preference Ordering

At any instance during the negotiation, one party is offering a list of choices to the other party. The issue is whether or not to require that the list be sorted based on end-user preferences? If sorted, the process is guaranteed to meet the end-user preferences faster. This is because, the negotiation could first determine the common choices that are preferred before spending time on less preferred options. To enable this we require that the calling application specify the ordering by indicating the preference criteria. The PN uses the GEPS Preference Management service to sort the capability list using those criteria. The U/I is responsible to obtain preference orderings when such information is not stored in the GEPS preference files.

#### 5.4.5 Soliciting Input

The issue of when to ask the end-user for input is a complicated one. Probably there are no two applications that share the exact policy. Nonetheless we want to enable minimum conformance by implementing a user interface that is consistent and yet allows flexible application design. To do this, we have specified the U/I as an Interface with minimum functionality requirements. The calling application is required to implement the U/I and can therefore enforce its own interface.

#### 5.5 Error Handling and Reporting

Several types of errors may occur while the PN is processing the input tokens. There are several ways to handle errors. First, a signal corresponding to the error is sent to the parent application upon occurrence of the error. Second, an exception is thrown upon an occurrence of an error and caught by the application that later consults an exception table to perform the recovery. Third, error related information is included within the token, as a status code; an application can read the status code from the token object and invoke a handler for that error type. Errors and their corresponding recovery steps are stored in a static error handler table, which is visible to all parts of the application and payment framework. Common errors that might occur include:

- Requested payment method is not available.
- Payment method is available, but no payment capability is installed which supports it.

- Payment method is available, but no available payment protocol supports it.
- Payment method, capability, and protocol are supported, but payment instrument is not accepted.
- The syntax of the token is malformed.
- The end-user performs abort, cancel, or retry operations.
- Other runtime errors that may occur.

When the application receives signal of a kind or an exception of a type, then it has to perform the sequence of steps specified in the global error table to recover from the error.

## 6 Security Consideration

There are two types of possible security attacks: denial of service attacks by corrupting the messages exchanged between the two parties and disclosure of capabilities resulting from eavesdropping the communications. Using a secure channel, it is possible to curtail these attacks. However, a malicious party can still cause the denial of service by interrupting the flow of messages between the two parties.

If the confidentiality of the payment capabilities is of paramount importance, then the parties could establish a secure channel over which they could conduct their communications. This way, the negotiation occurs over the secure channel and is not subject to eavesdropping.

## 7 Related Work

The payment negotiation framework presented in this paper supports a variety of negotiation and selection scenarios required by applications today. To the best of our knowledge it is the only work in this area addressing the complete range of issues including generic framework, common user interface, transport independence, end-user preferences, and generic programming interface. We intend to implement the service presented here as part of the Generic Electronic Payment Services [GEPS] framework. The UI and IOP components provide the common user interface and transport independence. The GEPS Preference Management service maintains the end-users' preferences and provides them for use by the Payment Method Negotiation service. While we have not specified a complete set of APIs in this paper, we have indicated the main set of required functions.

There is only one other related work that attempts to address part of the payment negotiation space. That is

the work performed in the Joint Electronic Payment Initiative [JEPI] project sponsored jointly by CommerceNet Consortium and the World Wide Web Consortium. We participated in JEPI during the first six month of this year as a cofounder and member of the design team. We made sure that our work builds on top and over the work already started by that team. The JEPI team is primarily concerned with specifying the syntax of the negotiation messages between the parties. They use UPP and PEP to form the messages and transport them. Our work uses that syntax but does more by building a supportive framework around it.

An implementation of the Payment Method Negotiation service has begun at the E-CO System project. You can access our [WebSite] for additional information about the project.

## 8 Conclusion

The excitement surrounding electronic commerce has created a lot of creative solutions for electronic payments. Transacting peers require a solid framework with which to negotiate and select among these choices. We believe that our Payment Method Negotiation service provides such a framework. Our service supports all phases of negotiation as specified in Section 4.1. Along with the GEPS, these services represent a complete set using which one can implement any application using electronic payments. Only time will tell if the implementation of all the functionality is possible. However, based on our early findings and the result of our implementation so far, all indications are in favor of such achievements.

We are addressing the technical challenges presented by the introduction of a variety of electronic payment solutions. However, there are other non-technical hurdles one has to overcome in order to successfully advocate the MN framework. These hurdles are related to the business acceptance of such a framework and the incorporation of such a framework into industry products. Currently, most businesses engage in competitive developments that prevents them from adopting open frameworks such as that advocated within this paper. In addition, perhaps it is too early in the development stages of the electronic payment industry to know enough what the future will hold and what framework is the right all encompassing and open model. We are positioning our work to address some of these issues, but only time will tell if we can achieve the industry acceptance required to make the framework ubiquitous.

## Acknowledgments

The author would like to acknowledge the dedication and team work of the E-CO System project members without whom the implementations and many aspects of this work would not have been possible. They are Rajkumar Narayanaswamy, James Galvin, Andrea Stirrup, and Nick Zhang.

In writing this document we reviewed the ideas presented in existing work. These include the Protocol Extension Protocol (PEP), Universal Payment Preamble (UPP) work, as well as any material available from the mailing lists of the Joint Electronic Payment Initiative (JEPI).

In closing, many thanks belong to Donald Eastlake of CyberCash. The author worked closely with him while in the JEPI Design Team and has profited from many of his insights and fruitful discussions.

## References

[WebSite] See the E-CO System Project website at <http://eco.eit.com>.

[GEPS] Alireza Bahreman, "Generic Electronic Payment Services: Framework and Functional Specification." Proceedings of the Second USENIX Electronic Commerce Workshop, November 1996.

[NegReq] Alireza Bahreman, "Electronic payment negotiation mechanism: Requirements document." VeriFone, January 9, 1996. Available at: <http://eco.eit.com/negotiation/NR.html>.

[PEP] Rohit Khare, "HTTP/1.2 Extension Protocol." Internet Draft <draft-ietf-http-pep-00>, August 1996.

[UPP] Donald E. Eastlake 3rd, "Universal Payment Preamble." Internet-Draft <draft-eastlake-universal-payment-03.txt>, August 1996.

[JEPI] See <http://www.w3.org/pub/WWW/Payments> and [www.commerce.net/work/taskforces/payments/jepi.html](http://www.commerce.net/work/taskforces/payments/jepi.html).

## Contact Information

You can reach the authors at:

Alireza Bahreman & Rajkumar Narayanaswamy  
bahreman@eit.com & rajkumar@eit.com  
EIT/VeriFone, 530 Lytton Avenue  
Palo Alto, CA 94301-1539, USA  
TEL: +1 415-617-9730; FAX: +1 415-617-9746

## Appendix A - Payment Method Negotiation Service Resources, Draft 0.1

Please note that this is a work in progress and therefore, the specifications might change.

```
class NegotiationContext {
    • Buyer/Seller Profile Information
    • TransactionDetails
    • Payment Capability List
    • Additional Payment Information
    • Negotiation Loop Counter
    • Utility functions such as:
        create_negotiation_context()
        update_negotiation_context()
        delete_negotiation_context()
        get_peer_information()
        get_negotiation_loop_count()
}

class NegotiationToken {
    • Negotiation Status Code
    • TransactionDetails
    • Sender/Receiver Profile Information
    • Payment Capability List
    • Additional Payment Information
    • Utility functions such as:
        get_payment_capability()
        get_capability_list()
}

class PartyProfile {
    • Hashtable of name, value pairs with
      information about the following:
    • name and identification details
    • affiliation
    • residence country
    • personal information
    • other relevant information
    • Utility functions
}

class TransactionDetails {
    • Transaction Handler
    • Price/Amount Details
    • Business Information
    • Miscellaneous parameters
    • Utility functions
}
```

## Appendix B - Blueprint for Classes and Functions in GEPS Payment Method Negotiation Service, Draft Version 0.1

Please note that this is a work in progress and therefore, the specifications might change.

```
abstract class IOProcessor {      // implement subclasses for each different transports
    public abstract TransportToken to_external_form();
    public abstract NegotiationToken to_internal_form();
}

public class PaymentNegotiator {          // different version for different policies
    public NegotiationToken begin_negotiation_context() {
        // similar to offer_negotiation_context except no capability list is formed
        // and output token type is Inquiry
    }

    public NegotiationToken offer_negotiation_context() {
        // Input: Caller preferred negotiation order and preference criteria for sorting
        // Output: NegotiationToken of type Request, Status, and NegotiationContext
        //           (if there already, the NegotiationContext is updated)

        - Get list of capabilities based on caller preferred negotiation order
          (i.e., payment method first, then payment protocol, and so on)
          use GEPS Capability Management that probes all payment capabilities installed
        - Sort them using caller's preference criteria and GEPS Preference Management
        - If NegotiationContext exists update it, otherwise create and update
        - Create new token holding the capability list with type Request
        - If successful, set status as Continue and return token
        - Otherwise, set status and token type to Failed
    }

    public NegotiationToken process_negotiation_context() {
        // Input: Token of type Inquiry, Request, or Intermediate
        //         Caller preferred negotiation order and preference criteria for sorting
        // Output: NegotiationToken of type Request, Status, updated NegotiationContext

        - If token of different type, return error
        - Get list of payment capabilities from the input token and NegotiationContext
        - Use GEPS Capability Management to validate all new offers
        - Reject any capability which does not meet needs or has been rejected before
        - Update NegotiationContext with new offers
        - Get list of new capabilities based on caller preferred negotiation order
          (i.e., payment method first, then payment protocol, and so on)
        - Create list of new offers and rejected offers and updates capabilities
        - Sort the list using caller's preference criteria and GEPS Preference
          Management if desired
        - Create token of type Intermediate that includes the new capability list
        - Return the token
    }

    public NegotiationToken finalize_negotiation() {
        // similar to process_negotiation_context except output token is of type Final
        // and input token is of type Intermediate
    }

    public NegotiationToken accept_negotiation () {
        // Input: Token of type Intermediate, or Final
        //         Caller preferred negotiation order and preference criteria for sorting
        // Output: NegotiationToken of type Selected, Status, updated NegotiationContext

        - If token of different type, return error
        - Get list of payment capabilities from the input token and NegotiationContext
        - Use GEPS Capability Management to validate all new offers
        - Reject any capability which does not meet needs or has been rejected before
        - Update NegotiationContext with new offers
        - Get list of new capabilities based on caller preferred negotiation order
          (i.e., payment method first, then payment protocol, and so on)
        - Create list of new offers and rejected offers and updates capabilities
        - Sort the list using caller's preference criteria and GEPS Preference
    }
}
```

```

        Management if desired
        - Use GEPS Preference Management to select one of the payment capabilities
        - If succeeded, return token of type Selected including the payment capability
        - If failed, throw an exception and set status and token type to Failed
    }

}

public interface UserInterface { // Uses Device Objects to interact with the end-user

    // For manually selecting a list of choices
    public PaymentBagArray get_choices(PaymentBagArray list) {
        DeviceHolder dh = new DeviceHolder(list, "SelectionDevice");
        dh.waitForResult();
        return dh.getAnswer();
    }

    // To ask the end-user to select a specific capability
    public PaymentBag select(PaymentBagArray list) {
        DeviceHolder cd = new DeviceHolder(list, "CheckboxDevice");
        cd.waitForResult();
        return cd.getAnswer();
    }

    // To ask the end-user to fill in the details manually
    public PaymentBag fill_details() {
        DeviceHolder fill = new DeviceHolder(null, "FillDevice");
        fill.waitForResult();
        return (PaymentBag) fill.getAnswer();
    }

    // To ask the end-user to confirm the current selection or proceedings of the
    // negotiation
    public String confirm(String question) {
        DeviceHolder dh = new DeviceHolder(question, "ConfirmationDevice");
        dh.waitForResult();
        return (String) dh.getAnswer();
    }

    // display some message to the end-user
    public void display_message(String mesg) {
        DeviceHolder dd = new DeviceHolder(mesg, "DisplayDevice");
        dd.waitForResult();
        return;
    }

    // add to the visual event viewer of the process
    public void add_system_messages(String mesg) {
        messages_area.add(mesg);
    }
}

```